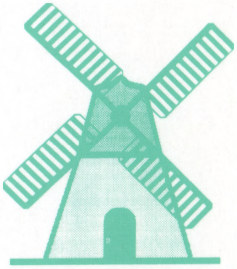
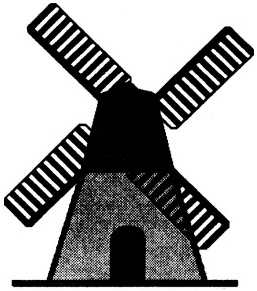
A+L AG

AMIGA OBERON



Amiga Oberon Compiler



Handbuch

© Copyright 1990 by

Fridtjof Siebert
Nobileweg 67
D-7000 Stuttgart 40

Diese Dokumentation ist urheberrechtlich geschützt. Die dadurch begründeten Rechte, besonders die der Übersetzung, des Nachdrucks, der Funksendung, der Wiedergabe auf photomechanischem oder ähnlichem Wege, der Speicherung und Auswertung in Datenverarbeitungsanlagen, bleiben, auch bei Verwertung von Teilen der Dokumentation, dem Autor vorbehalten.

Zur Beachtung: Die A+L AG und der Autor lehnen jede Art von Garantien ab, welche die Tauglichkeit der Software für einen bestimmten Zweck versprechen. Wir haften nicht für Fehler in dieser Dokumentation und auch nicht für Schäden, die durch den Gebrauch der Dokumentation, deren Inhalt oder der Software direkt oder indirekt entstehen.

Wir behalten uns vor, jederzeit Änderungen an dieser Dokumentation oder der Software vorzunehmen, ohne jeden Anwender davon zu benachrichtigen.

Herstellung und Vertrieb:

A+L AG
Däderiz 61
Ch-2540 Grenchen

Die Wiedergabe von Warenzeichen erfolgt ohne Gewährleistung der freien Verwendbarkeit.

Kundenunterstützung

Lesen Sie diese Information, bevor Sie das Siegel des Diskettenumschlags brechen.

Das Siegel des Diskettenumschlags zu brechen, bedeutet Akzeptieren der Bedingungen des A+L-Lizenzvertrages.

Die A+L AG und deren Partner helfen Ihnen, Ihr Programm optimal einzusetzen. Wir empfehlen Ihnen, die folgenden Informationen über Registration, Unterstützung und Ihre Rechte und Pflichten als Kunde durchzulesen.

Die A+L AG und deren Partner (dort, wo Sie das Produkt gekauft haben) bieten Ihnen telefonische Unterstützung durch einen Kundendienst. Egal wie kompliziert Ihre Frage oder Ihr Problem ist, wir versuchen Ihnen rasch zu helfen. Bevor Sie jedoch anrufen, vergewissern Sie sich, daß Sie Ihre Registrationskarte/ Ihren Lizenzvertrag eingeschickt haben. Andernfalls ist die A+L AG davon entbunden, Ihnen Auskunft zu geben.

Falls Probleme mit Ihrem Programm auftauchen, gehen Sie wie folgt vor:

1. Versuchen Sie, das Problem mit dem Handbuch zu lösen.

2. Rufen Sie A+L oder Ihren Händler zwischen 9 und 17 Uhr an. Sie sollten dann Ihre Programm-Seriennummer zur Hand haben.

Der Autor steht persönlich für Nachfragen nicht zur Verfügung. Probleme, die auf Fehler in der Software zurückzuführen sind, werden jedoch schnellstmöglich an den Autor weitergeleitet.

Beschränkte Garantie

A+L AG garantiert, daß:

1. das Material, Disketten und Dokumentation nicht beschädigt sind.

2. das Programm ordnungsgemäß auf die Disketten kopiert ist.

3. die Dokumentation vollständig ist und alle Informationen enthält, die A+L AG für die Bedienung des Programms für erforderlich hält.

4. das Programm im Prinzip so funktioniert, wie es im Handbuch beschrieben ist.

Falls Sie uns innert von 30 Tagen nach Auslieferung Fehler schriftlich melden oder uns defektes Material zustellen, steht Ihnen im Rahmen dieser beschränkten Garantie das Recht zu, Disketten oder Software ersetzt zu bekommen. Legen Sie der schriftlichen Fehlermeldung das entsprechende Rückporto in Ihrer Währung bei. Das Recht auf Ersatz besteht nur, wenn wir im Besitz Ihres unterschriebenen Lizenzvertrages sind. Senden Sie diesen sowie allfällige Fehlermeldungen oder defekte Disketten an Ihren Händler oder an folgende Adresse:

A+L AG
Däderiz 61
CH-2540 Grenchen
Schweiz

A+L AG haftet nicht für direkte oder indirekte Schäden.

A+L AG lehnt jede Art von Haftung oder stillschweigender Garantie ab, insbesondere, daß sich die Produkte von A+L AG für einen ganz bestimmten Zweck eignen. A+L AG beschränkt ihre

Garantie auf das Ersetzen defekter Disketten und Programme.

Entsprechend obenstehender Angaben tragen Sie als Anwender die Verantwortung, mit Ihrem Programm sorgfältig umzugehen, und damit auch die Kosten mutwilliger Beschädigungen und Fehler außerhalb der oben genannten Beschränkungen.

Anwenderlizenz

A+L AG behält sich alle Rechte für ihre Programme vor. Jedes Programm fällt unter diesen Lizenzvertrag. Ein entsprechendes Vergehen wird geahndet.

1. Erlaubter Gebrauch:

Das Programm darf nur auf einem einzigen Computer mit einem einzigen Bildschirm verwendet werden. Sie dürfen eine oder mehrere Kopien dieses Programms herstellen, damit Sie über ausreichend Kopiedisketten verfügen, falls Ihre Arbeitsdisketten zerstört werden. Dazu dürfen Sie eine und nur eine Kopie des Programms auf Ihrer Harddisk machen.

2. Unerlaubter Gebrauch:

Ohne ausdrückliche schriftliche Bewilligung der A+L AG dürfen Sie folgendes nicht tun:

- Das Programm in Verbindung mit mehr als einem Computer gleichzeitig verwenden.

- Das Programm, Kopien davon oder dessen Dokumentation an jemand anderen verkaufen.

- Erstellen von Kopien des Programms, der Dokumentation oder von Disketten, ausgenommen derjenigen Kopien, die in diesem Lizenzvertrag bewilligt sind.

- Die Verwendung des Programms in einem Netzwerk, einem Time-Sharing System, einem interaktiven Fernsehnetzwerk, einem Mehrprozessorensystem oder einem Mehrplatzsystem, falls nicht eine spezielle Lizenz von A+L vorliegt, beziehungsweise jeder einzelne Benutzer über eine eigene Lizenz mit A+L verfügt.

- Veränderungen am Programm vornehmen.

- Das Übertragen des Programms oder das Bewilligen einer Unterlizenz, Vermietung oder das Übertragen weiterer Rechte auf andere.

- Eine wörtliche oder übertragene Übersetzung des Programms herstellen.

- Anpassen des Programms an eine andere Hardware.

- Durchführen telefonischer oder elektronischer Datenübertragung des Programms.

- Vertrieb oder Vermietung des Programms an andere auf dauernder oder auch nur vorübergehender Basis.

- Das Programm oder damit erzeugte Programme oder Daten direkt oder indirekt militärisch nutzen.

Beendigung der Lizenz:

Die Lizenz gilt als beendet, wenn Sie alle Disketten, Dokumente und Kopien aller Art zerstören. Diese Lizenz fällt ebenfalls dahin, wenn Sie gegen eine der oben genannten Bedingungen verstoßen. Sie sind in einem solchen Fall verpflichtet, alle Ihre Disketten, Dokumente und Kopien aller Art zu vernichten.

Vorwort

Vor Ihnen liegt ein leistungsfähiger Compiler für eine der modernsten objektorientierten Sprachen. Oberon ist die neueste Entwicklung von Prof. Dr. Niklaus Wirth an der ETH Zürich. Diese Sprache entstand dort innerhalb eines Projekts, das sich das Ziel setzte, die Leistungsfähigkeit der Sprache Modula-II zu vergrößern und gleichzeitig ihre Komplexität zu verringern. Dazu wurde Oberon um das Konzept der Typerweiterung ergänzt. Dadurch wird objektorientiertes Programmieren ähnlich wie in Smalltalk oder C++ möglich, während jedoch die gewohnte Notation aus Modula-II und Pascal größtenteils beibehalten wird.

Dieser Compiler hat einige herausragende Fähigkeiten, die von anderen Compilern kaum erreicht werden:

- Der Compiler ist durch das Single-Pass-Konzept sehr schnell. Dennoch erzeugt er durch eine sorgfältige Auswahl an Optimierungen kurzen und schnellen Code.

- Die Objektdateien sind BLink-kompatibel. Dadurch sind sie sehr flexibel einsetzbar und die Einbindung von Routinen aus anderen Sprachen, wie Assembler und C, wird leicht möglich.

- Oberonprogramme werden optimierend gelinkt. Es kommen also keine Prozeduren, die niemals aufgerufen werden, in das ausführbare Programm, so daß dieses meist recht kurz wird. Ein 'leeres' Oberonprogramm ist zum Beispiel weniger als ein halbes Kilobyte lang.

- Durch das Setzen einer Compileroption ist es möglich, residentfähige, reentrante Programme zu erzeugen. Der Compiler und alle in Oberon geschriebenen mitgelieferten Programme sind residentfähig und reentrant.

- Programme, die mit diesem Compiler entwickelt wurden, sind sehr sicher im Laufzeitverhalten, da der Compiler massiv Überprüfungscode einsetzt, um möglichst alle Fehler abzufangen. Fehler führen in den meisten Fällen nicht zur "Guru-Meditation", sondern werden durch das Laufzeitsystem abgefangen.

- Um problemlos auf das Amiga-Betriebssystem und alle Resourcen zugreifen zu können, wurde die Sprache um Amiga-spezifische Typen ergänzt.

- Es stehen viele Bibliotheksmodule zur Verfügung, und es werden ständig neue Module geschrieben, die dann als Zusatzpakete erhältlich sein werden oder auf Public Domain Disktten erscheinen.

Dieser Compiler ist also ein ideales Werkzeug, um effektiv kleine und große Programme zu entwickeln. Durch die objektorientierten Eigenschaften kann eine einmal aufgebaute Modulbibliothek leicht für neue Projekte verwendet und die früher definierten Objekte können leicht an die neuen Aufgaben angepasst werden.

Der Amiga Modula und Oberon Klub Stuttgart (AMOK):

Um mit einem Compiler effektiv arbeiten zu können ist es von Vorteil, wenn man auf eine große Modulbibliothek zurückgreifen kann, da man sich dann eine Menge Entwicklungsarbeit ersparen kann. Zudem können Beispielprogramme vielen bei der Lösung von Problemen helfen.

Der Amiga Modula und Oberon Klub Stuttgart (AMOK) hat sich zum Ziel gesetzt, die Verbreitung der Sprachen Oberon und Modula zu unterstützen. Diese Sprachen haben den großen Vorteil, daß größere Programmprojekte durch Verwendung einer Modulbibliothek sehr viel einfacher realisierbar werden. AMOK ist ein Zusammenschluß von 6 Programmierern, die dies im Sommer 1988 erkannt haben und so die AMOK PD-Diskettenserie gegründet haben. Sie enthielt zunächst nur Programme und Module der Klubmitglieder. Unterdessen haben jedoch auch viele außenstehende Programmierer ihre Module und Programme freigegeben und uns zur Veröffentlichung überlassen. Dabei legt AMOK großen Wert auf die Qualität und Korrektheit (nicht unbedingt die Komplexität) der Module und veröffentlicht nur einen Teil des eingeschickten Materials.

Unterdessen hat sich eine mehrere Megabyte umfassende Bibliothek von Modula- und Oberon-Modulen angesammelt, die jedem zur Verfügung steht. Für viele Probleme können Lösungen auf den AMOK Disketten gefunden werden.

Um diese Idee auch weiter bestehen zu lassen, sind wir natürlich auf die Einsendung von Modulen und Programmen angewiesen. Wer also ein Modul geschrieben hat, von dem er denkt, daß es auch anderen von Nutzen sein kann, sollte es allen zur Verfügung stellen. Dadurch macht man sich nicht nur einen guten Namen in der Programmiererszene, sondern kann auch Kontakte zu anderen Programmierern knüpfen und so von der Veröffentlichung profitieren. Natürlich ist auch die

Veröffentlichung von sogenannter Shareware möglich, die dem Autor einen kleinen Nebenverdienst einbringen kann.

Die AMOK-Disketten können bei allen guten PD-Vertrieben und bei der A+L AG bezogen werden. Die Klubmitglieder freuen sich immer auf neue Programme und Module. AMOK-Mitglieder sind:

Nicolas Benezan
Postwiesenstr. 2
7000 Stuttgart 60

Kai Bolay
Hoffmannstr. 168
7250 Leonberg

Pit Burkhardt
Stettiner-Str. 25
7030 Böblingen

Michael Frieß
Mühlhaldenweg 16
7035 Waldenbuch

Bernd Kirschner
Gottlob-Grotz-Str. 24
7120 Bietigheim-Bissingen

Fridtjof Siebert
Nobileweg 67
7000 Stuttgart 40

Dabei sollten die Einsendungen möglichst auf alle Mitglieder verteilt werden, um den Arbeits- und Zeitaufwand des einzelnen gering zu halten. Wir sind alles 'nebenberufliche' AMOKler, die ihre Freizeit

und damit einen Teil der Zeit, in der sie sonst programmieren würden, für AMOK opfern. Wir geben keine telefonischen Auskünfte.

Wir können nicht garantieren, daß wir jeden Brief beantworten. Diejenigen Briefe werden meist bevorzugt, die ausreichend Rückporto enthalten. Da wir unsere Disketten nicht kommerziell vertreiben, sind wir auf Spenden angewiesen.

Literatur

[nw:aud]:

Niklaus Wirth:
Algorithmen und Datenstrukturen mit Modula-2
G.B. Teubner Verlag Stuttgart

[nw:or]:

Niklaus Wirth
Revised Oberon Report
ETH, Institut für Informatik
Fachgruppe Computer-Systeme
CH-8092 Zürich

[rb:agb]:

Das Amiga Guru Buch
Ralph Babel
Falkenweg 3
D-6204 Taunusstein

[cbm:airm]:

Amiga Intuition Reference Manual
Commodore Business Machines, Inc.
Addison Wesley

[cbm:ld]:

Amiga Rom Kernel Manual: Libraries & Devices
Commodore Business Machines, Inc.
Addison Wesley

[cbm:exec]:

Amiga Rom Kernel Manual: Exec
Commodore Business Machines, Inc.
Addison Wesley

Inhalt

1. Das erste Oberon-Programm

2. Installation

Installation auf Harddisk 1
Installation mit 2 Diskettenlaufwerken . . . 3
Installation mit 1 freien Diskettenlaufwerk . . 3

3. Der Editor OEd

Starten von OEd 1
Die Tastenbelegung 2
Die Maus 4
Die Menüs 5

4. Der Compiler Oberon

Die Namen der verwendeten Dateien . . . 1
Das Projekt-Konzept 2
Start des Compilers 2
Arbeitsweise des Compilers 5
Aufruf mit Batch-Datei 6

5. Linken mit OLink und BLink

Benutzung von OLink vom CLI 1
Benutzung von OLink von der Workbench . . 3
Arbeitsweise von OLink 3
Beispiel 4

6. Fehler anzeigen mit OErr

7. Das Tool ModToDef

Aufruf vom CLI 1
Aufruf von der Workbench 2
Arbeitsweise von ModToDef 2

8. Besonderheiten des Compilers

Parameter von HALT() 1
Speicherverwaltung 1
Neue Schlüsselwörter 2
Neue Standardbezeichner 4
Strukturierte Konstanten 5
Das Modul SYSTEM 6
Stringkonstanten 9
Reservierte Wörter und Operatoren 11
Standardbezeichner 12
Typgrößen 12
Einschränkungen 13
Das Laufzeitsystem 14
Compileroptionen 15
Die verschiedenen Speichermodelle . . . 20

9. Unterschiede zwischen Oberon und Modula-2

Neue Fähigkeiten von Oberon 1
Fähigkeiten, die gegenüber Modula-2 fehlen . . 7

10. Schnittstellen zu Libraries und Assemblercode

Definition von absoluten Variablen 1
Deklaration und Aufruf von Library-Prozeduren . 1
Zugriff auf Variablen und Prozeduren von
Assemblerroutinen 3

11. Wie erzeuge ich schnellen Code?

Speichermodelle 1
Variablen 1
Strukturanweisungen 2
Libraryaufrufe 3
Typen 3

12. Der erzeugte Code

Aufbau von Oberonprogrammen 1
Aufruf des Open- und Close-Codes 2
Zugriff auf Variablen 2
Zugriff auf String-, Feld- und Recordkonstanten . 3
Aufbau des Stapelspeichers bei globalen Prozeduren. 4
Aufbau des Stapelspeichers bei lokalen Prozeduren . 5
Format der offenen Feldparameter 6
Der erzeugte Überprüfungscode 6

13. Die Modulbiliothek

Arguments 1
ASCII 3
AVL 4
Beep 10
Break 11
BreakRq 12
Conversions 13
Display 15
FileReq 27
FileSystem 28
io 34
Lists 37
LongRealConversions 39
LongRealInOut 41

Mouse 42
NoGuru 43
NoGuruRq 45
OberonLib 46
Random 50
RealConversions 51
RealInOut 53
Requests 54
SecureDos 56
Strings 57

14. Die Amiga Interface Module

ARP 1
Audio 2
BootBlock 3
ClipBoard 4
Console 5
DiskFont 7
Dos 9
Exec 19
ExecSupport 32
Expansion 33
GamePort 37
Graphics 38
Hardware 58
Icon 68
Input 70
InputEvent 71
Intuition 73
KeyBoard 96
KeyMap 97
Layers 99
MathFFP 101
MathIEEEDoubBas 102

MathIEEEDoubTrans 104
MathTrans 105
Narrator 106
Parallel 108
Printer 110
PrtBase 114
Resources 117
SCSI 121
Serial 122
Timer 124
TrackDisk 125
Translator 127
Workbench 128

15. Das Make-Utility OMake

Aufruf von OMake 1
Start von der Workbench 1
Start vom CLI 1
Arbeitsweise von OMake 2

16. Der Library-Linker LibLink

Installation 1
Start von LibLink 1
Wichtige Hinweise 5

Anhang A: Amiga Oberon Version 2.0

Änderungen am Compiler 1
Änderungen am Editor 12
Die Modulbibliothek 15
Die Interface-Module 16

Index

1. Das erste Oberon-Programm

Dieses Kapitel ist für all diejenigen, die es nicht abwarten können, diese Anleitung vollständig zu lesen, bevor sie ihr erstes Programm schreiben. Wer gleich mit einem kleinen Programm anfangen möchte, sollte hier weiterlesen. Geduldigere können dieses Kapitel überspringen und in Ruhe mit der Installation (2. Kapitel) beginnen.

Wer zwei Diskettenlaufwerke besitzt, sollte die erste und die zweite Oberon-Diskette einlegen. Wer nur ein Laufwerk hat, muß im folgenden öfters die Disketten wechseln.

Zunächst muß QuickInstall von der ersten Diskette ausgeführt werden. QuickInstall kann einfach durch Doppelklick von der Workbench gestartet werden. Es führt den Befehl "ASSIGN OBERON: Oberon!:" aus und kopiert die arp.library (V1.3) in das LIBS-Directory der Workbench-Diskette. Es sollten daher noch etwa 17K Speicher auf der Workbench zur Verfügung stehen.

Nun kann der Editor OEd von der ersten Diskette durch einen Doppelklick aus der Workbench gestartet werden. Wer schon einmal mit einem Texteditor gearbeitet hat, wird in der Bedienung kaum Probleme haben. Nun kann ein Programm eingegeben werden, wie z.B. folgendes 'Hello World'-Demo:

```
MODULE Hello;

IMPORT io;

BEGIN
  io.WriteString("Hello World!"); io.WriteLn;
END Hello.
```

Dieses Programm sollte nun gespeichert werden. Dazu wählt man im Projekt-Menü den Menüpunkt 'Save As' und gibt anstelle des

"unnamed" den Namen "RAM:Hello.mod" ein. Durch Klick auf 'OK' wird der Text gespeichert. Nun kann der Editor durch einen Klick auf das CLOSE-Gadget verlassen werden.

Von der Workbench aus sollte man sich nun den Inhalt der RAM-Disk ansehen. Dort müßte ein Texticon mit dem Namen Hello.mod erscheinen. Das Windmühlen-Icon mit dem Namen 'OBERON' auf der ersten Oberon-Diskette ist der Compiler. Wird dieser einmal angeklickt und der Text in der RAM-Disk bei gedrückter Shift-Taste doppelgeklickt, beginnt der Compiler seine Arbeit und übersetzt den Text.

Um die erzeugten Dateien zu sehen, muß das Fenster der RAM-Disk geschlossen und wieder geöffnet werden. Es enthält nun zusätzlich die Dateien "Hello.sym" und "Hello.obj". Letztere wird einfach durch einen Doppelklick in ein ausführbares Programm umgewandelt.

Nochmaliges Schließen und Öffnen des RAM-Disk-Fensters zeigt, daß 3 neue Dateien erzeugt wurden: "Hello.lnk", "Hello.wth" und "Hello". Nur die letzte interessiert uns im Augenblick. Sie ist das ausführbare Programm und kann durch Doppelklick gestartet werden.

Wie man sieht, ist das Erstellen von Programmen sehr einfach. Die Funktion der erzeugten Dateien wird in den folgenden Kapiteln noch genau erklärt.

2. Installation:

Um diesen Compiler sinnvoll benutzen zu können, sollte man einen Amiga mit mindestens 1 MByte Speicher haben. Mehr Speicher kann jedoch niemals schaden. 2 Diskettenstationen werden empfohlen, eine ist jedoch ausreichend. Wer jemals mit einer Harddisk gearbeitet hat, wird diese auch nie wieder missen wollen.

Vor der Installation des Compilers sollten zunächst Sicherheitskopien der Disketten hergestellt und die Originaldisketten an einen sicheren Platz gebracht werden.

Installation auf Harddisk:

Zunächst sollte ein neues Directory erzeugt werden, das sinnvollerweise den Namen Oberon bekommt. In dieses Directory müssen folgende Dateien und Directories von der ersten Oberon-Diskette kopiert werden:

- BLink (hat kein Icon)
- Fehler-Meldungen
- ModToDef
- Oberon
- OEd
- OErr
- OLink
- Path
- Das Directory Icons mitsamt seinem Inhalt

Zudem muß die arp.library aus dem libs-Directory dieser Diskette nach LIBS: kopiert werden, wenn man ARP noch nicht installiert hat. Die s:Startup-Sequence muß um folgende 2 Zeilen erweitert werden:

```
ASSIGN OBERON: DHx:Oberon
PATH OBERON: ADD
```

Die erste Zeile teilt dem Compiler, Linker etc. mit, wo die Oberon-Dateien zu suchen sind. Die zweite Zeile ermöglicht das Starten des Compilers vom CLI aus, ohne den Pfad jedesmal angeben zu müssen. Diese Zeile muß dann auch in die Datei s:CLI-Startup übernommen werden.

Wer ausreichend Speicher hat, kann den Compiler und den Linker resident machen. Dann müssen die Programme beim Starten vom CLI aus nicht mehr nachgeladen werden. Wer die ARP-Shell installiert hat, erweitert die Startup-Sequence um folgende Zeilen:

```
ARES OBERON
ARES OLINK
```

Wer Arp nicht verwendet, kann auch die Befehle der Workbench 1.3 benutzen:

```
RESIDENT OBERON:OBERON
RESIDENT OBERON:OLINK
```

Nun müssen noch die Objekt- und Symboldateien von der zweiten Diskette in dieses Directory kopiert werden. Dazu können einfach die beiden Directories "obj" und "sym" in das Oberon-Directory kopiert werden.

Zum Schluß muß dem Compiler noch mitgeteilt werden, wo er die Objekt- und Symboldateien suchen soll. Dies steht in der Textdatei OBERON:Path. Es reicht, wenn diese Datei eine Zeile mit dem Inhalt "OBERON:" enthält. Wer die Objekt- und Symboldateien auch in anderen Directories speichert, sollte deren Pfade auch in die Pfaddatei eintragen. Die Datei kann mit jedem ASCII-Editor (z.Bsp. OEd) editiert werden. Jede Zeile sollte einen Pfad enthalten. Beispiel:

```
OBERON:
DF0:
```

Installation mit 2 Diskettenlaufwerken:

Für jemanden, der zwei freie Diskettenlaufwerke besitzt, ist die Installation sehr einfach. Er erweitert lediglich seine Startup-Sequence um folgende 2 Zeilen:

```
ASSIGN OBERON: Oberon_Disk1:
PATH OBERON: ADD
```

Die erste Zeile kann dabei weggelassen werden, wenn man die erste Oberon-Diskette mittels "RELABEL Oberon_Disk1: OBERON" oder mit Rename von der Workbench umbenennt. Die zweite Zeile ist nur nötig, wenn man vom CLI aus arbeiten möchte.

Die Datei OBERON:Path muß wie bei der Installation auf Harddisk alle Pfade enthalten, in denen sich die Objekt und Symboldateien befinden. Dies ist zunächst nur die zweite Diskette. Es reicht hier also die Zeile "Oberon_Disk2:".

Installation mit einem freien Diskettenlaufwerk:

Etwas schwieriger wird die Installation, wenn nur eine Diskettenstation zur Verfügung steht, da man entweder keine zweite hat, oder gerne die Workbench-Diskette ständig in einem Laufwerk hätte. Dazu muß man sich zunächst etwas Platz auf der ersten Oberon-Diskette schaffen. Es können, bis auf die folgenden, alle Dateien gelöscht werden:

- BLink (hat kein Icon)
- Fehler-Meldungen
- ModToDef (kann evtl. auch gelöscht werden)
- Oberon
- OEd
- OErr
- OLink

- Path
- Directory Icons mitsamt seinem Inhalt (nur nötig, wenn man von der Workbench arbeiten möchte)

Auf der Diskette sollten nun 2 Unterdirectories mit den Namen "obj" und "sym" erzeugt werden. In diese Directories sollten alle oft benötigten Objekt- und Symboldateien der zweiten Diskette kopiert werden. Es zeigt sich normalerweise erst mit der Zeit, welche Dateien besonders oft benötigt werden. Dies hängt stark davon ab, was für Programme man schreibt. Folgende Objekt- und Symboldateien werden im Allgemeinen seltener benutzt:

ARP	AVL	Beep
BootBlock	Clipboard	DiskFont
Display	Expansion	GamePort
Icon	Keyboard	Layers
Lists	Mouse	Narrator
Parallel	Printer	Resources
Serial	TrackDisk	Translator

Die entsprechenden Dateien mit der Endung .obj, .objs oder .sym brauchen also nicht auf die Diskette kopiert zu werden. Wird das kleine Datenmodell nicht verwendet, sind die Dateien mit der Endung .objs nicht nötig. Zusätzlich können alle Icons der Dateien entfernt werden, da sie normalerweise nicht benötigt werden.

Wie bei der Installation mit zwei Laufwerken sollte die Startup-Sequence um folgende 2 Zeilen erweitert werden:

```
ASSIGN OBERON: Oberon_Disk1:
PATH OBERON: ADD
```

Die erste Zeile kann auch hier weggelassen werden, wenn man die Diskette in "OBERON" umbenennt und die zweite Zeile ist nur nötig, wenn man mit dem CLI arbeiten möchte.

In die Pfad-Datei sollte als erster Pfad "OBERON:" eingetragen werden. Da sich in diesem Pfad jedoch nicht alle Objekt- und Symboldateien befinden, da manche gelöscht wurden, sollte als zweiter Pfad noch die zweite Oberon-Diskette angegeben werden, also "Oberon_Disk2:". Wenn die gelöschten Dateien dann wirklich einmal benötigt werden, ist ein Diskettenwechsel nötig.

3. Der Editor OEd:

Dieser Editor wurde speziell für diesen und mit diesem Compiler entwickelt und unterstützt besonders die Entwicklung von Oberonprogram en. Er kann leicht mit der Maus bedient werden, erlaubt das gleichzeitige Bearbeiten mehrere Texte, enthält einen Oberon-Syntaxchecker, erlaubt das Starten von Compiler, Linker und compiliertem Programm und kann die Fehlermeldungen des Compilers anzeigen.

Starten von OEd:

Gestartet wird OEd einfach durch Doppelklick auf sein Icon oder durch Eingabe von "OEd" im CLI. Werden Argumente übergeben, oder wurden von der Workbench zuvor Texte angeklickt und OEd mit gedrückter Shift-Taste doppelgeklickt, dann werden die angewählten Texte automatisch in jeweils ein Fenster geladen.

Beim Start vom CLI aus können eine ganze Reihe an Argumente übergeben werden.

Aufruf:

OEd {-{i|s|t#|x#|y#|w#|h#|d#|l}} {<Text>}

Dabei steht das '#' für eine Dezimalahl. Die Optionen haben fogende Bedeutung:

-i	Icons erzeugen
-s	OEd soll eigenen Screen öffnen
-t#	Tabulatorbreite setzen. Voreingestellt ist 2
-x#,-y#	x- und y-Position der OEd-Fenster
-w#,-h#	Breite und Höhe der OEd-Fenster
-d#	Tiefe des OEd-Screens. Voreingestellt ist 2
-l	Interlace-Screen öffnen

Beispiel:

```
OEd -sy11d1 Test.mod
```

Hier öffnet OEd das Textfenster mit dem Quelltext "Test.mod" auf einem zweifarbigen Screen direkt unter der Screen-Titelzeile.

Wird OEd von der Workbench gestartet, können diese Voreinstellungen mit Hilfe der Tooltypes übergeben werden. Dazu klickt man auf das OEd-Icon und wählt 'Info' aus dem Workbench-Menü. Im Tooltype-Gadget können nun neue Werte für die Fenstergröße, Tabulatorweite etc. angegeben werden.

Die Titelzeile des OEd-Fenster dient als Editor-Statuszeile. Sie zeigt wichtige Informationen, wie Fehlermeldungen etc. an. Normalerweise enthält sie in etwa folgendes:

```
col: 024  line: 0007 of 0219 * file: Test.mod
```

Dabei ist die Zahl hinter "col:" die Spalte (hier 24) und die Zahl hinter "line:" die Zeile (hier 7), in der sich der Cursor gerade befindet. Hinter "of" steht die Gesamtzahl der Zeilen im Text (hier 219). Das Sternchen dahinter erscheint immer dann, wenn an dem Text seit dem letzten Speichern oder Laden eine Änderung vorgenommen wurde. Der Text, der "file:" folgt, ist der Name der gerade bearbeiteten Datei (Test.mod).

Die Tastenbelegung:

Cursortasten (die Tasten mit den Pfeilen):

Wie Sie sich vielliecht schon gedacht haben, kann man mit diesen Tasten den Cursor im Text bewegen.

Zusammen mit der Alt-Taste kann man mit den horizontalen Pfeilen

wortweise durch den Text springen. Alt zusammen mit den vertikalen Pfeilen springt seitenweise durch den Text.

Mit Shift springt man jeweils an den Anfang oder das Ende, d.h. mit den horizontalen Pfeilen an den Zeilenanfang bzw. an das Zeilenende und mit den vertikalen Pfeilen an den Textanfang bzw. an das Textende.

Backspace (die Taste rechts oben mit dem Pfeil nach links):

Mit dieser Taste kann das Zeichen links vom Cursor gelöscht werden. Am Zeilenanfang hat sie keine Funktion. Zusammen mit Alt wird das Wort links vom Cursor gelöscht.

Del:

Mit Del wird das Zeichen unter dem Cursor gelöscht. Shift zusammen mit Del gedrückt löscht die Zeile, in dem sich der Cursor befindet. Control (Ctrl) und Del löschen alles, was in der Zeile an und rechts von der Cursorposition steht. Mit Alt und Del kann das Wort rechts vom Cursor gelöscht werden.

Return:

Mit der Return-Taste wird eine Zeile unterhalb der Zeile, in der sich der Cursor befindet, eingefügt. Der Cursor wird in die neue Zeile unter das erste Zeichen der Zeile darüber gesetzt.

TAB:

Springt zur nächsten, oder bei gedrückter Shift-Taste zur vorigen Tabulator-Marke. Der Tabulatorabstand beträgt normalerweise 2. Er kann beim Starten von OEd mit "-t#" gesetzt werden, wobei # durch die gewünschte Ziffer ersetzt werden muß.

Control+'S':

Zerteilt (*splits*) die Zeile in der sich der Cursor befindet in 2 Zeilen. Dabei kommen alle Zeichen ab der Cursorposition in die neue Zeile.

Control+'J':

Vereinigt (*joins*) die Zeile, in der sich der Cursor befindet, mit der Zeile darunter. Die beiden Zeilen werden aneinandergehängt und die untere Zeile wird gelöscht. Dies ist das Gegenstück zu Split (Control+'S').

ESC:

Mit der Escape-Taste kann das aktive OEd-Fenster auf die Größe eines Icons verkleinert werden (iconified). Dadurch kann man sich Platz schaffen, wenn man dieses Fenster eine Weile nicht benötigt, es jedoch nicht schließen möchte. Nochmaliges Drücken von ESC vergrößert das Fenster wieder auf die ursprüngliche Größe.

Amiga + 'A' .. 'Z':

Die Menüpunkte (s.u.) können auch durch Drücken der rechten Amiga-Taste zusammen mit Buchstaben ausgewählt werden.

Die Maus:

Mit der Maus kann der Cursor an eine bestimmte Position im Text gesetzt werden, indem man die gewünschte Position anfährt und kurz den linken Mausknopf drückt.

Um mit der Maus einen Block zu markieren, läßt man die Maustaste gedrückt und fährt soweit nach oben oder unten, bis der gewünschte Bereich markiert ist. Nun kann die Maustaste losgelassen werden. Genaures über die Bearbeitungsmöglichkeiten von Blöcken siehe unten.

Die Menüs:

Die Menüs von OEd bieten eine Vielzahl von Möglichkeiten, den Text zu bearbeiten, sowie ein paar spezielle Funktionen zur Behandlung von Oberon-Programmen, wie Syntaxüberprüfung, Starten des Compilers und Anzeigen der Fehlermeldungen.

Das Projekt Menü:

Load (Amiga + 'L'):

Öffnet einen File-Requester zur Eingabe des Namens einer neuen Datei. Wurde die alte Datei verändert und noch nicht gespeichert, erfolgt noch eine Sicherheitsabfrage, bevor die veränderte Datei gelöscht wird.

New Window (Amiga + 'W'):

Öffnet ein neues (leeres) Fenster. In dieses Fenster kann dann ein weiterer Text geladen werden.

Save (Amiga + 'S'):

Speichert den Text unter dem Namen, der in der Statuszeile steht.

Save As (Amiga + 'V'):

Öffnet einen File-Requester, damit ein Dateiname eingegeben werden kann, unter dem der gerade bearbeitete Text dann gespeichert wird.

Save block (Amiga + 'O'):

Speichert den mit der Maus oder dem Block-Menü definierten Block in einer Datei. Um den Dateinamen einzulesen, wird ein File-Requester geöffnet.

Insert File (Amiga + 'I'):

Fügt eine Datei über der Zeile, in der sich der Cursor befindet, in den Text ein. Zusammen mit dem Menüpunkt 'Save block' hat man so eine Cut- und Paste- Funktion für Blöcke.

Quit (Amiga + 'Q'):

Schließt das aktive Fenster. Wurde der bearbeitete Text verändert und nicht gespeichert, findet noch eine Sicherheitsabfrage statt. Andere Fenster, die mit 'New Window' oder durch zusätzliche Argumente beim Start geöffnet wurden, bleiben weiterhin offen.

About (Amiga + 'J'):

Öffnet einen Requester mit Informationen über die Version und den Autor des Editors.

Das Search Menü:

Find (Amiga + 'F'):

Öffnet ein Fenster mit einem String-Gadget. Dort soll ein Suchstring eingegeben werden. Nach der Beendigung der Eingabe mit Return wird der Text ausgehend von der Cursorposition nach dem String durchsucht. Der Cursor wird an den Anfang des gefunden Strings gesetzt. Ist die Suche erfolglos, blitzt der Bildschirm kurz auf.

Next (Amiga + 'N'):

Next sucht nach dem nächsten Vorkommen des Strings, der bei Find eingegeben wurde, und setzt den Cursor an dessen Anfang. Bei erfolgloser Suche blitzt der Bildschirm kurz.

Previous (Amiga + 'P'):

Durchsucht den Text rückwärts nach dem String der bei Find eingegeben wurde und setzt den Cursor an dessen Anfang.

FindRep (Amiga + 'G'):

FindRep liest zunächst wie Find einen String ein. Danach wird ein zweiter String eingelesen, durch den der erste ersetzt werden soll. Der zweite String kann leer sein, um den ersten String aus dem Text zu löschen.

Nach der Eingabe sucht FindRep nach dem ersten Vorkommen des ersten Strings (Such-String) ausgehend von der Cursorposition und ersetzt ihn dort durch den zweiten Strings (Ersatz-String).

NextRep (Amiga + 'R'):

Sucht von der Cursorposition ausgehend nach dem nächsten Vorkommen des Such-Strings (siehe FindRep) und ersetzt ihn durch den Ersatz-String.

PrevRep (Amiga + 'H'):

Sucht ausgehend von der Cursorposition rückwärts nach dem Such-String (siehe FindRep) und ersetzt ihn durch den Ersatz-String.

Das Block Menü:

Begin (Amiga + 'B'):

Löscht die Markierung des evtl. zuvor definierten Blocks und setzt den Blockanfang auf die Zeile, in der sich der Cursor befindet.

End (Amiga + 'E'):

Setzt die Blockende-Markierung auf die Zeile, in der sich der Cursor befindet. Der nun neu definierte Block wird hervorgehoben. Im folgenden wird dieser Block der markierte Block genannt. Ein Block kann auch leicht mit der Maus markiert werden (s.o.).

Copy (Amiga + 'Y'):

Kopiert den markierten Block vor die Zeile, in der sich der Cursor befindet. Der ursprüngliche Block bleibt unverändert.

Move (Amiga + 'M'):

Verschiebt den markierten Block vor die Zeile, in der sich der Cursor befindet. Der Block wird dabei an seiner ursprünglichen Position gelöscht.

Delete (Amiga + 'D'):

Löscht den markierten Block aus dem Text. Diese Funktion kann nicht rückgängig gemacht werden!

TAB right (Ctrl + 'R'):

Verschiebt den markierten Block um eine Tabulatorweite (normalerweise 2) nach rechts.

TAB left (Ctrl + 'L'):

Verschiebt den markierten Block um eine Tabulatorweite nach links.

Unmark (Amiga + 'U'):

Löscht die Blockmarkierung. Es ändert sich dabei am Blockinhalt nichts. Er wird lediglich nicht mehr hervorgehoben gezeichnet und es existiert kein markierter Block mehr.

Das Oberon Menü:

Die in diesem Menü enthaltenen Funktionen erleichtern das Erstellen von Oberonprogrammen. Sie sollten beim Editieren anderer Texte nicht verwendet werden.

Parse (Amiga + 'A'):

Der Text wird 'geparst', d.h. es wird geprüft, ob der Text ein Element der Oberon-Grammatik ist, die der Oberon-Compiler versteht (inklusive der Zusätze zum Original-Oberon). Enthält das Programm Syntaxfehler, wird der Cursor an die fehlerhafte Stelle gesetzt und eine kurze Fehlermeldung in der Statuszeile angezeigt.

Das Parsen ist sehr viel schneller als das Compilieren eines Programms. Es sollte daher jedesmal vor der Compilation aufgerufen werden. So können grobe Fehler, wie vergessene END's oder falsch gesetzte Klammern, die den Compiler oft zu Folgefehlern verleiten, schnell gefunden werden.

Logische Fehler, wie inkompatible Typen etc., werden beim Parsen nicht erkannt.

Compile (Amiga + 'C'):

Startet den Oberon-Compiler mit dem Namen des gerade bearbeiteten Textes als Argument. Der Compiler übernimmt den Text direkt aus dem Speicher, so deß der Text vor der Compilation nicht auf Diskette gespeichert werden muß. Während der Compilation kann der Text nicht weiter editiert werden.

Damit diese und die nächsten vier Funktionen korrekt arbeiten können, müssen sich die CLI-Kommandos CD, STACK, PATH und RUN im C:-Directory befinden. Wer ausreichend Speicher hat, sollte diese Befehle resident laden.

Außerdem sollte das Verzeichnis T: existieren, damit OEd eine temporäre Datei erzeugen kann.

Link (Amiga + 'K'):

Startet OLink, so daß das zuvor compilierte Programm gelinkt wird.

Execute (Amiga + 'X'):

Startet das ausführbare (gelinkte) Programm. Dabei wird ein Start von dem CLI ohne Argumente simuliert.

Next Error (Amiga + 'T'):

Wurden bei der Compilation Fehler gefunden, können diese mit 'Next Error' angesehen werden. 'Next Error' zeigt immer den nächsten Fehler, der sich rechts oder unterhalb des Cursors befindet. Der Fehler wird in der Statuszeile angezeigt. Die Meldung 'No more Errors' erscheint, wenn es unterhalb des Cursor keine weiteren Fehler gibt.

First Error (Amiga + 'Z'):

Springt an die Position des ersten Fehlers im Text und zeigt ihn in der Statuszeile an. Wurden bei der Compilation keine Fehler gefunden, wird die Meldung 'No Errors' ausgegeben.

Das Options Menü:

Hier können die Editor-, Compiler- und Linkeroptionen eingestellt werden:

Insert:

Ist *Insert* angewählt, befindet sich der Editor im Einfüge-, sonst im Überschreib-Modus.

Icons:

Ist diese Option angewählt, werden Icons erzeugt.

StackChk, OvflChk, RangeChk, CaseChk, ReturnChk, NilChk und TypeChk:

Diese Optionen bewirken, daß der entsprechende Überprüfungscode beim Compilieren erzeugt wird. Genaueres dazu steht in Kapitel 8.

SmallCode und SmallData:

Wählen das Kleine Code- bzw. Daten-Modell. Genaueres steht auch hierzu in Kapitel 8.

4. Der Compiler Oberon:

Das Programm 'Oberon' ist der eigentliche Compiler. Er erledigt die meiste Arbeit: Er erhält die mit OEd oder einem anderen ASCII-Editor geschrieben Quelltexte als Eingabe. Daraus erzeugt er dann eine Symboldatei, die Information über alle exportierten Objekte des compilierten Moduls enthält, und eine Objektdatei, die den erzeugten Maschinencode, also das eigentliche Programm, enthält.

Wurden bei der Compilation Fehler im Quelltext entdeckt, wird keine Symbol- und keine Objektdatei erzeugt. Stattdessen schreibt der Compiler Informationen über die Positionen und die Nummern der Fehlermeldungen in eine Datei.

Die Namen der verwendeten Dateien:

Der Typ von bestimmten Dateien wird durch das Anhängen einer Endung an den Dateinamen bestimmt. Bei diesem Oberon-System werden folgende Dateinamen verwendet:

Endung	Dateityp
'.mod'	Modul- und Programm-Quelltexte.
'.modE'	Fehlerdatei nach Compilation eines fehlerhaften Moduls
'.def'	Mit dem Tool 'ModToDef' erzeugtes Definitionsmodul
'.obj'	Objektdatei, von Compiler erzeugt
'.objs'	Objektdatei bei Verwendung des kleinen Datenmodells
'.sym'	Symboldatei, von Compiler erzeugt
'.lnk'	Link Datei, von OLink erzeugt
'.wth'	WITH-Datei für BLink, von OLink erzeugt
ohne Endung	Ausführbares Programm nach dem Linken mit OLink / BLink

Das Projekt-Konzept:

Beim Entwickeln größerer Programme, die aus mehreren Modulen bestehen, kann die große Anzahl an Dateien ein einzelnes Verzeichnis schnell unübersichtlich machen.

Um eine bessere Übersicht über ein Programmprojekt zu behalten, empfiehlt es sich, drei Unterverzeichnisse mit den Namen 'txt', 'obj' und 'sym' mit dem CLI-Befehl CreateDir zu erzeugen. Wer von der Workbench arbeitet, kann sich das Empty-Directory seiner Workbench-Diskette dreimal kopieren und entsprechend umbenennen.

Die Quelltexte sollten nun in das 'txt'-Verzeichnis geschrieben werden. Bei der Compilation holt sie sich der Compiler automatisch von dort. Entsprechend werden die Objekt- und Symboldateien vom Compiler automatisch in die Unterverzeichnisse geschrieben bzw. von dort gelesen.

Start des Compilers:

Aufruf aus dem Editor:

Im Editor kann der Compiler einfach durch Anwahl des Menüpunktes 'Compile' gestartet werden. Dabei muß der Quelltext zuvor gespeichert worden sein.

Aufruf vom CLI:

Wer mit dem CLI arbeitet, sollte das aktive Verzeichnis mit dem CLI-Befehl 'CD' zunächst auf das Verzeichnis setzen, das die Quelltexte oder das 'txt'-Unterverzeichnis enthält.

Der Compiler benötigt normalerweise mehr als 4 KByte Stapelspei-

cher. Der Stapel sollte daher mit dem CLI-Commando 'stack' auf 10000 bis 20000 Bytes gesetzt werden.

Nun kann der Compiler mit folgender Syntax gestartet werden:

Oberon { [-svbcrntmdi] <Quelltext> }

Quelltext ist der Text, der compiliert werden soll. Dabei kann die Endung '.mod' weggelassen werden. Sie wird dann automatisch angehängt. Befindet sich der Quelltext in dem Unterverzeichnis mit dem Namen 'txt', wird er automatisch von dort geholt. Es kann also z. B. statt der Zeile

Oberon txt/Test.mod

einfach

Oberon Test

geschrieben werden.

Der Compiler kann auch mit mehreren Quelltexten gestartet werden. Er übersetzt sie dann nacheinander.

Die Optionen ('[-svbcrntmdi]') beziehen sich auf alle Quelltexte, die rechts von ihnen stehen. Sie haben folgende Wirkung:

Option	Wirkung
'-s'	Stackkontrolle ausschalten
'-v'	Überlaufskontrolle ausschalten
'-b'	Bereichskontrolle ausschalten
'-c'	Caseindex-kontrolle ausschalten
'-r'	Returnkontrolle ausschalten
'-n'	Nil-Zeiger-Kontrolle ausschalten
'-t'	Typkontrolle ausschalten

'-m' Kleines Code-Modell verwenden
'-d' Kleines Daten-Modell verwenen
'-i' Icons erzeugen

Genaueres zu den verschiedenen Arten an Überprüfungscode und den verschiedenen Speichermodellen steht im Kapitel 'Besonderheiten des Compilers'.

Die Optionen beziehen sich auf alle Quelltexte, die hinter der Option stehen. Beispiel:

```
Oberon -s Prog1 -b Prog2
```

Hier wird Prog1 ohne Stackkontrolle und Prog2 ohne Stackkontrolle und ohne Bereichskontrolle compiliert.

Wurde der Compiler ohne Parameter gestartet, gibt er den Prompt 'in>' aus und wartet auf die Eingabe des Quelltextnamens. Verlassen wird der Compiler dann einfach durch Drücken von Return.

Aufruf von der Workbench:

Wird der Compiler von der Workbench gestartet, müssen die Quelltexte bei gedrückter Shift-Taste angeklickt und danach der Compiler doppelgeklickt werden.

Von der Workbench aus sollte der Compiler nicht ohne Parameter aufgerufen werden, da er dann die Objekt- und Symboldateien nicht in das Verzeichnis schreibt, in dem sich die Quelltexte befinden, sondern in das Verzeichnis, in dem sich der Compiler selbst befindet. Dies ist meist nicht erwünscht.

Beim Start von der Workbench aus erzeugt der Compiler automatisch Icons.

Arbeitsweise des Compilers:

Der Compiler zeigt an, welche Dateien er einliest und welche er erzeugt. Dabei schreibt er ' - <Dateiname>' beim Einlesen einer Datei und ' + <Dateiname>' beim Erzeugen einer Datei.

Während der Compilation werden zunächst die Symboldateien der importierten Module geladen. Für jedes Modul wird ' - XXX.sym' ausgeben. Ist die Importliste abgearbeitet, wird Code erzeugt. Dabei wird für jedes erzeugte Kilobyte ein Punkt ausgeben.

Wird im Programm mit REAL- oder LONGREAL-Zahlen gerechnet, werden eventuell noch die Symboldateien der Mathematikmodule nachgeladen.

Ist der Text compiliert, überprüft der Compiler, ob eine neue Symboldatei erzeugt werden muß. Dazu liest er die eventuell schon existierende Symboldatei ein und vergleicht sie mit der, die er erzeugen würde. Wenn Unterschiede gefunden wurden, wird eine neue Symboldatei erzeugt und als 'XXX.sym' oder als 'sym/XXX.sym', wenn das 'sym'-Verzeichnis existiert, gespeichert.

Nun wird der Code in etwa 2-8 zusätzlichen Passes optimiert. Dabei gibt der Compiler 'optimizing ...' und die Zahl der wegoptimierten Bytes aus. Der fertig optimierte Code wird dann im Amiga-Objektformat als 'XXX.obj' oder als 'obj/XXX.obj', wenn das 'obj'-Verzeichnis existiert, gespeichert.

Zum Schluß wird noch die Größe der Code-, Daten- und Variablenbereiche (BSS) ausgegeben.

Werden während der Compilation Fehler gefunden, so werden diese in die Datei XXX.modE gespeichert. Die Fehler können dann mit den Programmen OEd oder OErr angesehen werden.

Beispiel einer Compilation:

```
in> Demo
 - Demo.mod
 - OBERON:sym/Graphics.sym
 - OBERON:sym/Hardware.sym
 - OBERON:sym/Exec.sym
 - OBERON:sym/Intuition.sym
 - OBERON:sym/KeyMap.sym
 - OBERON:sym/Timer.sym
 - OBERON:sym/InputEvent.sym
 - OBERON:sym/Dos.sym
 - OBERON:sym/OberonLib.sym
.. - sym/Demo.sym

optimizing ...  80
optimizing ...  10
optimizing ...   0
 + obj/Demo.obj
 CODE: 2368  DATA:    0  BSS:  1760
```

Hier wurde das Programm 'Demo.mod' compiliert.

Aufruf mit Batch-Datei:

Um mehrere Module auf einmal zu compilieren, kann man sich eine Batch-Datei schreiben. Dies ist eine einfache Textdatei, in der in jeder Zeile der Name eines zu compilierenden Moduls steht. Diese Datei kann z.B. unter dem Namen 'bat' gespeichert werden.

Um die Module in der Batchdatei zu compilieren, ruft man den Compiler vom CLI folgendermaßen auf:

```
Oberon <bat
```

Er wird also parameterlos gestartet. Der Inhalt der Batchdatei wird als

Eingabestrom verwendet und übergibt dem Compiler automatisch die Modulnamen.

5. Linken mit OLink und BLink:

Die vom Compiler erzeugten Objektdateien sind Standard-Amiga-Objektdateien. Diese Dateien können daher mit den Standardlinkern ALink oder BLink zu einem ausführbaren Programm zusammengefügt werden.

Der Aufruf von BLink ist jedoch nicht ganz einfach: BLink kann nur vom CLI benutzt werden und verlangt die komplette Liste aller Objektdateien der vom Hauptmodul importierten Module. Dies ist oft viel Tipparbeit, und leicht wird eine Objektdatei vergessen.

OLink ist ein Programm, das BLink automatisch mit den richtigen Parametern startet. Zudem kann OLink auch von der Workbench benutzt werden und erzeugt Icons.

Benutzung von OLink vom CLI:

Aufruf:

OLink [-bsmdi] <Objektdatei> {OBJ <file>}

Dabei ist <Objektdatei> die Objektdatei des Hauptmoduls des Programms, das gelinkt werden soll. Ähnlich wie beim Compiler kann die Endung '.obj' bzw. '.objs' und der Pfad 'obj/' weggelassen werden. So kann z.B. statt

OLink obj/Test.obj

einfach

OLink Test

geschrieben werden.

Die Optionen ('[-bsmdi]') haben folgende Funktionen:

Option	Wirkung
'-b'	Verhindert, daß BLink gestartet wird.
'-s'	Die Optionen SMALLCODE und SMALLDATA werden beim Aufruf von BLink verwendet. Dadurch werden die Programme etwas kürzer. Der Nachteil ist, daß die Programme nur noch aus großen Programmblöcken bestehen, d.h. ein 100K großes Programm braucht in etwa 100K durchgehenden Speicher und kann evt. nicht mehr geladen werden, obwohl noch genügend Speicher vorhanden wäre, die einzelnen Speicherblöcke jedoch zu klein sind.
'-m'	Setzt die Option SMALLCODE beim Starten von BLink. Diese sollte bei Programmen, die mit dem kleinen Code-Modell compiliert wurden, gesetzt sein. Sonst werden die Programme länger und langsamer.
'-d'	Diese Option muß gesetzt werden, wenn mit dem kleinen Daten-Modell compiliert wurde, damit die Objektdateien mit der Endung '.objs' verwendet werden. Zudem muß Olink hier die Größe des Speichers bestimmen, den die globalen Variablen belegen.
'-i'	Icons erzeugen.

Hinter dem Wort 'OBJ' können noch zusätzliche Objektdateien angegeben werden, die z.B. in Assembler geschrieben sind und von einem der Oberonmodule verwendet werden. OLink kann diese Dateien normalerweise nicht selbst finden, da in den Oberon-Objektdateien nur die Namen der importierten Oberonmodule stehen.

Wird OLink ohne Parameter gestartet, erwartet es ähnlich wie der Compiler die Eingabe eines Dateinamens.

Benutzung von OLink von der Workbench aus:

Von der Workbench aus kann OLink sehr leicht gestartet werden, indem man das Icon der Objektdatei des Hauptmoduls doppelklickt. Dazu muß sich OLink im logischen Verzeichnis 'OBERON:' befinden, was gewöhnlich der Fall ist.

Arbeitsweise von OLink:

Ist OLink gestartet, durchsucht es die Objektdatei nach Verweisen auf andere Module. Die Objektdateien dieser Module werden dann jeweils geladen und ebenfalls untersucht, bis keine Verweise auf neue Objektdateien mehr gefunden werden.

Aus den Namen der gefundenen Objektdateien wird nun eine sogenannte WITH-Datei erzeugt. Dies ist eine Textdatei, die BLink übergeben wird. Zusätzlich enthält sie den Namen des fertigen Programms und eventuell die Optionen 'SMALLCODE' und 'SMALLDATA'.

Beim Linken von Programmen, die mit dem kleinen Daten-Modell compiliert wurden, müssen zusätzlich noch die Größe und der Typ des Speichers der globalen Variablen bestimmt werden. Diese werden als Zuweisung zu Symbolen in die WITH-Datei geschrieben.

Nachdem alle Objektdateien gefunden wurden und die WITH-Datei erzeugt wurde, wird nun BLink gestartet und das ausführbare Programm erzeugt. BLink muß sich dazu im Verzeichnis OBERON: befinden. Nun kann das fertige Programm gestartet und ausgetestet werden.

OLink hat noch eine weitere Datei erzeugt, die bisher noch keine Rolle gepielt hat: Eine Text-Datei mit der Endung '.lnk'. Dies ist eine Script-Datei, die BLink mit der WITH-Datei als Parameter startet. Muß ein Programm, dessen Modulstruktur sich nicht verändert hat, d.h. dessen IMPORT-Liste gleichgeblieben ist, noch einmal gelinkt

werden, kann dies einfacher und schneller durch Ausführen der '.lnk'-Datei geschehen:

Von der Workbench reicht ein Doppelklick auf das Icon dieser Datei. Vom CLI muß 'Execute XYZ.lnk' oder wenn die Shell der Workbench 1.3 verwendet wird, einfach 'XYZ.lnk' eingegeben werden.

Bei Verwendung des kleinen Datenmodells sollte die '.lnk'-Datei nicht benützt werden, da sich die WITH-Datei auch bei gleicher Modulstruktur ändern kann. Hier ist es am besten, OLink bei jedem Linken zu starten.

Beispiel:

Wurde OLink mit 'OLink -s Demo.mod' gestartet, kann die WITH-Datei z.B. folgendermaßen aussehen:

```
FROM obj/Demo.obj
LIBRARY
    OBERON:obj/OberonLib.obj
    OBERON:obj/Dos.obj
    OBERON:obj/Intuition.obj
    OBERON:obj/Graphics.obj
TO Demo
SC SD
```

Dabei ist 'obj/Demo.obj' das Hauptmodul. Die Dateien, deren Namen unter 'LIBRARY' stehen, sind Bibliotheksmodule, die optimierend dazugelinkt werden.

Hinter 'TO' steht der Name des erzeugten Programms. 'SC' und 'SD' sind die Abkürzungen für die Optionen 'SMALLCODE' und 'SMALLDATA'.

Die erzeugte Link-Datei sieht folgendermaßen aus:

```
OBERON:BLink WITH Demo.wth
```

Sie ist, wie man sieht, sehr kurz. Wird diese Datei ausgeführt, so wird BLink direkt gestartet, ohne daß OLink noch einmal nach den Objektdateien suchen muß.

Die Ausgabe von OLink sieht folgendermaßen aus:

```
in> Demo
 - obj/Demo.obj
 - OBERON:obj/OberonLib.obj
 - OBERON:obj/Dos.obj
 - OBERON:obj/Intuition.obj
 - OBERON:obj/Graphics.obj
 + Demo.wth
 + Demo.lnk
Blink - Version 6.7 - 15 October 1986
Copyright © 1986 The Software Distillery.  All Rights Reserved.
235 Trillingham Lane, Cary NC 27511 - BBS: (919)-471-6436


BLINK Complete - Maximum code size = 5096 ($000013e8) bytes

tschüß!
```

Die von BLink ausgegebende 'Maximum code size' gibt an, wieviel Speicher das Programm benötigt, wenn es in den Computer geladen wird. Dieser Wert ist unabhängig davon, ob OLink mit der Option '-s' gestartet wurde oder nicht.

6. Fehler anzeigen mit OErr:

Traten bei der Compilation Fehler auf, können sie mit dem Hilfsprogramm OErr angesehen werden. Dies ist nötig, wenn man einen anderen Editor als OEd verwendet, der keine Oberon-Fehlermeldungen anzeigen kann.

Es werden jeweils die Zeilennummer und die Zeile angezeigt, in der der Fehler festgestellt wurde. Die Position wird mit einem Pfeil ('^') markiert. Darunter werden die Fehlernummern, gefolgt von den Fehlermeldungen, geschrieben.

Beispiel:

```
317:  Me := FindTask(NIL);
                  ^
      25: Bezeichner nicht definiert
     121: Bezeichner sollte Prozedur sein
                      ^
      70: Zu viele Parameter
                        ^
      76: Zugewiesener Wert hat falschen Typ
```

Hier wurde in Zeile 317 vergessen, daß FindTask ein von Exec importierter Bezeichner ist und nur qualifiziert verwendet werden darf.

7. Das Tool ModToDef:

In Oberon gibt es keine Definitionsmodule wie in Modula-2. Stattdessen ist jedes Modul eine einzige Textdatei, die die Funktion der Definition und der Implementation gleichzeitig übernimmt. Alle exportierten Objekte werden durch Sternchen hinter den Bezeichnern markiert. Dadurch wird das Schreiben von Modulen leichter und übersichtlicher.

Es gibt jedoch auch ein paar Vorteile von Definitionsmodulen, die man in Oberon nicht vermissen möchte:

- ein Definitionsmodul enthält die exportierten Bezeichner eines Modules sehr kompakt und übersichtlich.

- Wenn der Modulautor seine Module anderen zur Verfügung stellen möchte, seinen Quelltext jedoch nicht freigeben will, kann er mit dem Definitionsmodul leicht jedem die Benutzung des Moduls ermöglichen. Zudem müssen natürlich die Symbol- und Objektdateien mitgeliefert werden.

ModToDef ist ein Programm, daß aus einem Oberon-Modul ein Definitionsmodul erzeugt. Die Definitionen müssen also nicht wie in Modula-2 von Hand geschrieben werden. Sie werden einfach aus dem Quelltext erzeugt.

Aufruf vom CLI:

Aufruf:

ModToDef {<QuellText>}

Es können mehrere Definitionsmodule gleichzeitig erzeugt werden, indem mehrere Quelltexte angegeben werden. Die Endung '.mod' kann auch hier weggelassen werden.

Wird kein Quelltext als Parameter übergeben, wartet ModToDef wie der Compiler auf die Eingabe eines Quelltextnamens.

Aufruf von der Workbench aus:

Von der Workbench aus kann ModToDef wie der Compiler durch Anklicken des Quelltextes und Doppelklicken von ModToDef bei gedrückter Shift-Taste gestartet werden.

Arbeitsweise von ModToDef:

Zunächst wird der Quelltext eingelesen. Danach wird die Definition erzeugt und mit der Endung '.def' abgespeichert. Der Quelltext wird durchlaufen und nach exportierten Objekten durchsucht. Diese werden in die Definition geschrieben.

Definitionen beginnen mit dem Wort 'DEFINITION', damit man sie von Modulen unterscheiden kann.

Damit ModToDef die Importliste korrekt erzeugt, müssen alle Module, von denen Typen in der Definition von exportierten Bezeichnern verwendet werden, auch mit einem Sternchen markiert werden. Dies ist beim Compiler nicht nötig.

Die erzeugten Definitionsmodule werden von keinem Programm benötigt und können auch von keinem Programm bearbeitet werden. Sie werden nur für die Programmierer benötigt, die damit arbeiten und eine kurze Übersicht über den Inhalt von Modulen brauchen.

Beispiel: Aus folgendem Modul

```
MODULE GGT;
```

```
PROCEDURE GGT*(a,b: INTEGER): INTEGER;
BEGIN
  LOOP
    IF a=b THEN RETURN a
    ELSIF a>b THEN DEC(a,b) ELSE DEC(b,a) END;
  END;
END GGT;

END GGT.
```

wird diese Definition erzeugt:

```
DEFINITION GGT;

PROCEDURE GGT(a, b: INTEGER): INTEGER;

END GGT.
```

8. Besonderheiten des Compilers:

Hier werden die Unterschiede zum Oberon-Report [nw:or], Erweiterungen der Sprache und die Bedeutung der verschiedenen Compileroptionen beschrieben.

Parameter von HALT():

Der Parameter der Standardprozedur ist eine LONGINT-Konstante. Sie wird beim Programmstart vom CLI als Fehlercode zurückgegeben (siehe Amiga-DOS Handbuch unter FAILAT).

Dieser Wert sollte 0 sein, wenn ein Programm korrekt arbeiten konnte. Traten irgendwelche Fehler auf, sollte ein Wert zwischen 1 und 20 an HALT übergeben werden, je nach der Schwere des Fehlers.

Beispiel:

```
NEW(p); IF p=NIL THEN HALT(20) END;
```

Der Nachteil von HALT() ist, daß es normalerweise keine Fehlermeldung ausgibt, so daß man es nicht immer bemerkt, wenn ein Programm fehlgeschlagen ist.

Speicherverwaltung:

Diese Oberon-Implementation verwaltet den Speicher nicht über 'Garbage-Collection', so wie dies im Oberon-Report verlangt wird. Der Grund hierfür liegt im Betriebssystem Amiga-DOS, das den Speicher anders verwaltet.

Dieser Compiler verwaltet den Speicher wie in Pascal mit den Standardprozeduren NEW() und DISPOSE(). Dabei alloziert NEW(p) einen SYSTEM.SIZE(p^) großen Block und weist seine Adresse p zu. Ist nicht genügend Speicher vorhanden, wird p auf NIL gesetzt.

Der allozierte Speicher wird automatisch beim Beenden des Programms freigegeben, jedoch nicht schon während der Laufzeit. Werden große Speicherblocks nicht mehr benötigt, können sie mit der Standardprozedur DISPOSE(p) freigegeben werden. Dabei muß jedoch darauf geachtet werden, daß man danach nicht mehr auf diesen Speicher zugreift, da sonst Abstürze die Folge sein können.

Viele Oberonprogramme können trotz der anderen Speicherverwaltung auch unverändert auf dem Amiga laufen. Sie benötigen jedoch eventuell mehr Speicher.

Neue Schlüsselwörter STRUCT, BPOINTER, CLOSE, AND und NOT:

Um die Entwicklung von Oberonprogrammen zu erleichtern und den Zugriff auf Amiga-spezifische Datenstrukturen zu ermöglichen, wurde die Sprache um folgende reservierte Schlüsselworte erweitertert. Diese dürfen nicht als Bezeichnernamen verwendet werden.

STRUCT:

RECORDs in Oberon enthalten immer unsichtbare Daten, in denen Informationen über den Typ des Records erhalten sind. Dies wirkt sich störend aus, wenn man auf die Datenstrukturen von Modula-2- oder C-Programmen zugreifen möchte, die keine zusätzlichen Informationen enthalten.

STRUCT kann in Typdefinitionen verwendet werden, um Verbundtypen wie die RECORDs in Modula-II oder die Strukturen in C zu definieren. Die Syntax entspricht der RECORD-Definition, mit den Unterschieden, daß keine Basistypen angegeben werden dürfen und das Schlüsselwort RECORD durch STRUCT ersetzt wird.

Beispiel:

```
TYPE
  MInNodePtr = POINTER  TO  MinNode;
  MinList = STRUCT
    head : MinNodePtr;
    tail : MinNodePtr;
    tailPred : MinNodePtr;
  END;
```

Dieser Typ ist 12 Bytes groß. Würde statt STRUCT RECORD verwendet, wäre er 16 Bytes lang und enthielte zusätzlich ein Feld für den Typeguard (siehe [nw:or]).

STRUCT sollte nie dazu verwendet werden, Speicher zu sparen. Bei STRUCT geht die Möglichkeit, RECORDs zu erweitern, verloren. Sie können nur sehr viel unflexibler benutzt werden.

BPOINTER:

Die Dos Library des Amiga Betriebssystems, die in BCPL geschrieben ist, speichert Zeiger anders, als dies sonst im Amiga üblich ist. Daher müssen diese Zeiger gesondert behandelt werden.

Das Schlüsselwort BPOINTER kann gleich wie POINTER verwendet werden. Variablen des Typs "BPOINTER TO xyz" haben folgende Eigenschaften:

- Werden sie an LONGINT zugewiesen, so werden sie zuvor in normale Zeiger umgewandelt. Werden umgekehrt LONGINTs an sie zugewiesen, so werden diese vorher in BCPL-Zeiger umgewandelt.

- Werden BPOINTER dereferenziert (mit "^" oder automatisch mit "." oder "[..]"), so werden sie zuvor in normale Zeiger umgewandelt.

BPOINTER sollten nicht auf normale RECORDs zeigen, da Typtests nur mit normalen Zeigern möglich sind. Diese Zeiger sollten nur beim Zugriff auf DOS-Strukturen verwendet werden, dort wo dies unbedingt nötig ist. Überall sonst verlangsamen und verlängern BPOINTER den Code, ohne irgendwelche Vorteile zu bringen.

CLOSE:

CLOSE leitet eine Anweisungsfolge in einem Modul ein, die beim Beenden des Programms aufgerufen wird. Dort sollten alle verwendeten Resourcen freigegeben werden.

Beispiel:

```
MODULE abc;
  ...
BEGIN
  Window := OpenWin();
  IF Window=NIL THEN HALT(20) END;
CLOSE
  IF Window#NIL THEN CloseWin(Window) END;
END abc.
```

So ist sichergestellt, daß das Programm das Fenster wieder schließt, unabhängig davon, wie es beendet wurde.

AND und NOT:

Die Schlüsselworte AND und NOT sind Synonyme für die Operatoren '&' und '~'. Sie können verwendet werden, um boolesche Ausdrücke leichter lesbar zu machen.

Neue Standardbezeichner:

Die Liste der Standardbezeichner wurde um die drei Bezeichner

DISPOSE, SHORTSET und LONGSET erweitert. DISPOSE ist eine Standardprozedur, die im Unterkapitel 'Speicherverwaltung' genauer beschrieben wird.

SHORTSET und LONGSET sind Typbezeichner für SETs mit einer Größe von 8 bzw. 32 Bit. Um diese SETs in SET-Ausdrücken von normalen SETs zu unterschieden, muß der jeweilige SET-Typbezeichner wie in Modula-2 vor die geschweifte Klammer geschrieben werden.

Beispiel:

```
VAR s: LONGSET;
BEGIN
  s := LONGSET{19..29};
END;
```

Ansonsten können diese SETs wie gewöhnliche SETs behandelt werden.

Strukturierte Konstanten:

Diesem Compiler wurde die in ähnlichen Sprachen oft vermißte Fähigkeit gegeben, Record- und Feldkonstanten zu definieren. Dies sieht ähnlich wie ein Prozeduraufruf aus, wobei der Typbezeichner als Prozedurbezeichner dient und die Record- bzw. Feldelemente als Parameter übergeben werden.

Die Elemente werden in der Reihenfolge steigender Indizes (bei Feldern) oder in der Reihenfolge wie Typdefinition (bei Records) angegeben. Bei erweiterten Records müssen zunächst die Elemente der Basisrecords und dann der Erweiterungen angegeben werden. Sind die Record- oder Feldelemente selbst Records oder Felder, müssen deren Elemente einzeln angegeben werden.

Auf strukturierte Konstanten kann wie auf Variablen gleichen Typs zugegriffen werden, sie können jedoch nicht verändert werden.

Beispiel:

```
MODULE Test;

IMPORT I: Intuition;

CONST
  nw = I.NewWindow(0,0,500,200,1,0,LONGSET{},LONGSET{},
                   NIL,NIL,NIL,NIL,NIL,
                   0,0,0,0,{I.wbenchScreen});

VAR
  NW: I.NewWindow;
  w: I.WindowPtr;

BEGIN
  NW := nw;
  w := I.OpenWindow(NW);
  IF w#NIL THEN
    I.CloseWindow(w)
  END;
END Test.
```

Das Modul SYSTEM:

SYSTEM ist ein compilerinternes Modul, das eine Reihe von 'low-level' Prozeduren enthält, die direkte Manipulation von Registern und Daten erlauben. Diese Prozeduren sollten so selten wie nur möglich verwendet werden.

PROCEDURE ADR(x: ARRAY OF BYTE): LONGINT;

Das Ergebnis von ADR() ist die Adresse von x. x kann dabei von einem beliebigen Typ sein.

ADR() kann in Oberonprogrammen, die sauber geschrieben wurden und keine direkten Betriebssystemaufrufe enthalten, immer vermieden werden. So kann zum Beispiel in

```
VAR a: xyz;
BEGIN
  b := SYSTEM.ADR(a);
END;
```

ADR(a) vermieden werden, wenn man a gleich als Zeiger definiert:

```
VAR a: POINTER TO xyz;
BEGIN
  NEW(a); Requests.Assert(a#NIL,"Kein Speicher");
  b := a;
END;
```

Programme die SYSTEM.ADR() und DISPOSE() nicht verwenden, sind frei von sogenannten hängenden Referenzen, d.h. Zeigern auf Speicherbereiche, die keine gültigen Werte mehr enthalten. Dadurch können folgenschwere Fehler vermieden werden.

PROCEDURE SIZE(x: ARRAY OF BYTE / Type): Integer;

SIZE(x) ergibt den von x benötigten Speicher in Bytes. x kann dabei ein Typ oder eine Variable sein.

PROCEDURE VAL(T: Type; x: ARRAY OF BYTE): T;

Interpretiert x, als wenn es vom Typ T ist. Dadurch sind Typumwandlungen möglich. SIZE(T) und SIZE(x) müssen gleich sein.

PROCEDURE INIT(p: POINTER);

Diese Prozedur dient dazu, die Typeguards (siehe [nw:or]) in Records zu setzen. Dies geschieht normalerweise automatisch, wenn Speicher

mit NEW() alloziert wird. Wenn für RECORDs Speicher z.B. mit Exec.AllocMem() alloziert wird, müssen die Typeguards noch in das Record geschrieben werden, damit die Typ-Tests korrekt funktionieren.

Beispiel:

```
VAR p: POINTER TO RECORD ... END;
BEGIN
  p := MyAllocMem(SYSTEM.SIZE(p^));
  IF p=NIL THEN HALT(20) ELSE SYSTEM.INIT(p) END;
END;
```

PROCEDURE ROT(x: Integer / Set; n: INTEGER): Integer / Set;

Rotiert die Bits in x um n Positionen nach links, oder nach rechts für ein negatives n. x kann von einem beliebigen Integer- oder Set-Typ sein.

PROCEDURE LSH(x: Integer / Set; n: INTEGER): Integer / Set;

Verschiebt die Bits in x um n Positionen nach links, oder nach rechts für negative n. Dabei werden die freiwerdenden Bits mit Nullen gefüllt. Mit dieser Funktion können z.B. Zweierpotenzen berechnet werden. Das Ergebnis von LSH(1,n) ist vom Typ SHORTINT und ist der Wert 2 hoch n.

PROCEDURE REG(n: INTGER): LONGINT;

REG() liefert den Inhalt der Register des 68000er-Prozessors. n ist eine INTEGER-Konstante, die das Register auswählt. Dabei meinen Werte von 0 bis 7 die Register D0 bis D7. Die Werte 8 bis 15 ergeben den Inhalt der Register A0 bis A7.

PROCEDURE SETREG(n: INTEGER; x: ARRAY OF BYTE);

Kopiert den Wert x in das 68000er-Register mit der Nummer n (Nummer wie bei REG()). x darf höchstens 4 Bytes groß sein. Ist SIZE(x) nur 1 oder 2, so wird x lediglich in das unterste Byte bzw. Wort des Registers geschrieben.

PROCEDURE INLINE(x1,x2,x3,...: INTEGER);

INLINE fügt konstante Wörter direkt in den Code-Teil der erzeugten Objektdatei ein. Dadurch kann ein Programm kurze Maschinencodeprogramme oder Daten enthalten. INLINE() akzeptiert Werte, die zwischen -8000H und +0FFFFH liegen. Die Werte > 7FFFH werden wie vorzeichenlose INTEGERs behandelt.

Da der Compiler passiven Code entfernt, kann es bei Statements wie

```
LOOP EXIT; INLINE(...) END;
```

passieren, daß der INLINE-Code wieder entfernt wird.

Stringkonstanten:

Konstante Zeichenketten stehen immer automatisch an geraden Adressen und enden mit einem 0X. Zeichenketten können mit Anführungszeichen '"' oder Apostrophen "'" definiert werden.

Auf konstante Strings kann zeichenweise zugegriffen werden. Dies geschieht wie bei gewöhnlichen ARRAYs. Es können alle Zeichen einschließlich des 0X am Zeichenkettenende ausgelesen werden.

Beispiel:

```
PROCEDURE IntToHex(int: LONGINT;
            VAR hex: ARRAY OF CHAR; n: INTEGER);
CONST digits = "0123456789ABCDEF";
```

```
VAR x: LONGINT;
BEGIN
  hex[n] := 0X;
  WHILE n>0 DO
    DEC(n);
    x := int MOD 16;
    IF x<0 THEN INC(x,16) END;
    hex[n] := digits[x];
    int := int DIV 16;
  END;
END IntToHex;
```

Eine weitere Besonderheit sind Steuerzeichen, die mit Hilfe des Backslashes ("\") eingefügt werden können. Der Backslash wird von einem Buchstaben oder Zeichen gefolgt. Die Zeichen haben folgende Bedeutungen:

Zeichen	Bedeutung	
\n	Line-feed	0AX
\t	Tabulator	09X
\r	Carriage-return	0DX
\b	Backspace	08X
\f	Form-feed	0CX
\o	nul	00X
\e	Escape	1BX
\\	Backslash	5CX
\'	single quote	"'"
\"	double quote	'"'
\NNN	Das Zeichen mit der Octalzahl NNN	
\xHH	Das Zeichen mit der Hexzahl HH (0HHX)	
\[	'\e[' = Control Sequence Introducer (CSI)	

Das wohl wichtigste Zeichen ist LF ("\n"), das beim Aufruf von io.WriteString das gleiche bewirkt wie io.WriteLn. Mit Hilfe des CSI

("\[") kann die Schriftart leicht verändert werden. So schaltet z.B. "\[1m" auf Fettschrift, und "\[m" schaltet wieder zurück auf normale Schrift. Genaueres dazu steht in [cbm:ld].

Ein Backslash am Zeilenende bewirkt, daß die Zeichenkette in der nächsten Zeile fortgesetzt werden kann. Das Zeilentrennzeichen im Quelltext (LF) wird dann überlesen.

Alle anderen Zeichen hinter einem Backslash werden unverändert übernommen. Dies betrifft auch Steuerzeichen im Quelltext.

Zeichenketten dürfen nie auf ungeraden Adressen liegen. Dies ist wichtig, wenn mit Zeigern auf Zeichenketten gearbeitet wird und man den Zeiger mitten in einen Text zeigen läßt.

Reservierte Wörter und Operatoren:

Dieser Compiler kennt folgende Operatoren und reservierte Wörter, die nicht als Bezeichnernamen verwendet werden können:

"	#	&	'
(	)	*	+
,	-	.	..
/	:	:=	;
<	<=	=	>
>=	[	]	^
{	|	}	~
AND	ARRAY	BEGIN	BPOINTER
CASE	CLOSE	CONST	DIV
DO	ELSE	ELSIF	END
EXIT	IF	IMPORT	IN
IS	LOOP	MOD	MODULE
NOT	OF	OR	POINTER
PROCEDURE	RECORD	REPEAT	RETURN
STRUCT	THEN	TO	TYPE
UNTIL	VAR	WHILE	WITH

Standardbezeichner:

Folgende Bezeichner sind vordefiniert:

ABS	ASH	BOOLEAN	BYTE
CAP	CHAR	CHR	COPY
DEC	DISPOSE	ENTIER	EXCL
FALSE	HALT	INC	INCL
INTEGER	LEN	LONG	LONGINT
LONGREAL	LONGSET	MAX	MIN
NEW	NIL	ODD	ORD
REAL	SET	SHORT	SHORTINT
SHORTSET	TRUE		

Typgrößen:

Die Standardtypen können folgende Werte annehmen:

Typ	MIN(Typ)	MAX(Typ)	SIZE(Typ)
BOOLEAN	FALSE	TRUE	1
CHAR	0X	0FFX	1
BYTE	0	255	1
SHORTINT	-128	127	1
INTEGER	-32768	32767	2
LONGINT	-2147483648	2147483647	4
REAL	-9.223317*10^18	9.223317*10^18	4
LONGREAL	-10^308	10^308	8
SHORTSET	0	7	1
SET	0	15	2
LONGSET	0	31	4

Bei den Settypen beziehen sich MIN() und MAX() auf die maximale bzw. minimale Größe der Setelemente.

Einschränkungen:

Da ein (realer) Computer eine endliche Maschine ist, muß es bei den Programmen, die auf diesem Computer laufen sollen, gewisse Restriktionen geben. Diesen Einschränkungen unterliegt auch der Compiler. Zudem wurden noch weitere Einschränkungen gemacht, die keine größeren Nachteile bringen, die Programme jedoch beschleunigen, verkürzen und vereinfachen.

- maximale Codegröße je Modul 32768 Bytes (vor der Optimierung). Ein Modul, das diesen Umfang hat, sollte ohnehin wegen der besseren Übersichtlichkeit schon längst in zwei Module gespalten worden sein.

- maximaler Stringkonstantenbereich je Modul 32768 Bytes. Wenn dieser Bereich jemals erschöpft werden sollte, kann ein zweites Modul, das nur Strings enthält, geschrieben werden.

- maximale Länge von Bezeichnern 79 Zeichen.

- maximaler Bereich von globalen Variablen je Modul: 2 GByte. Um effizient auf alle Variablen zugreifen zu können, sollte der Variablenbereich kleiner als 32KByte sein.

- maximale Größe von offenen Feldparametern: 32KByte.

- maximale Größe des Stapelbereichs der aktiven Prozedur: 32KByte. D.h. die Größe der Prozedurparameter und der lokalen Variablen darf zusammen nie mehr als 32K sein. Eine Abhilfe ist das Arbeiten mit Zeigern, die dann mit NEW() alloziert werden, statt mit Recordvariablen.

- maximale Symboldateigröße: 7FFFFEH = 8388606 Bytes.

- maximale Anzahl Module je Programm: 32768.

- maximale Schachtelungstiefe stapelbarer Optionen: 64.

- maximal 256 RECORD-Definitionen je Modul, jedoch beliebig viele STRUCTs.

Das Laufzeitsystem:

Bei der Entwicklung der Laufzeitunterstützung wurde besonderer Wert auf Effizienz und Kürze gelegt. Dies wirkt sich jedoch nicht negativ auf die Sicherheit aus.

Das Standard-Laufzeitsystem befindet sich im Modul 'OberonLib'. Dieses Modul wird automatisch in alle Oberonprogramme eingebunden und übernimmt dort grundlegende Aufgaben, wie z.B. die Speicherverwaltung und das Rechnen mit LONGINT-Zahlen. Zudem fängt es Laufzeitfehler ab. Dies sind z.B. Überläufe beim Rechnen oder das Verwenden von Zeigern, die den Wert NIL enthalten. Solche Fehler führen in den seltensten Fällen zu Systemabstürzen. Sie werden normalerweise abgefangen. Dabei wird jedoch keine Fehlermeldung ausgegeben. Wurde ein Programm von der Workbench gestartet, kann man den Fehler nur daran bemerken, daß das Programm plötzlich beendet wird.

Im CLI hat man es etwas einfacher: ein fehlerhaftes Programm gibt -1 als Returncode zurück. Das Dos meldet sich dann mit der zwar wenig sagenden, aber immerhin vorhandenen Fehlermeldung 'Unable to load "xyz"'. Diese Meldung signalisiert einen schweren Fehler im Programm.

Damit Laufzeitfehler vielsagender angezeigt werden, sollten alle Programme, zumindest während der Entwicklungsphase, eines der Module 'NoGuru' oder 'NoGuruRq' importieren. Diese Module geben bei einem Laufzeitfehler die genaue Fehlerursache im CLI-Fenster bzw. in einem Requester aus. 'NoGuru' gibt zudem noch die Prozessorregister ('Dx' und 'Ax'), den Program-Counter ('pc'), das Statusregister ('sr') und evtl. noch weitere Informationen aus. Assembler-

freaks können damit, solange sich das fehlerhafte Programm noch im Speicher befindet, genauere Informationen über Fehlerort und Fehlerursache gewinnen.

Oberonprogramme können normalerweise nicht mit der Tastenkombination Control und 'C' abgebrochen werden. Dies ist jedoch bei Programmen, die länger laufen und nicht jederzeit beendet werden können, oft wichtig. Deshalb gibt es die Module 'Break' und 'BreakRq', die einen einfachen Abbruch mit dieser Tastenkombination erlauben. Sie brauchen, genauso wie die NoGuru-Module, nur in der Importliste erwähnt werden.

Compileroptionen:

Durch das Setzen oder Löschen bestimmter Optionen kann der Compiler veranlaßt werden, unterschiedlichen Code zu erzeugen, der aus Gründen der Effizienz oder beim Schreiben von bestimmten Programmteilen, wie z.B. Prozeduren, die im Interrupt ablaufen sollen, nötig ist.

Die Optionen, die Überprüfungscode ausschalten, sollten sparsam und nur an Stellen, von deren Korrektheit man sich überzeugt hat, verwendet werden.

Die Optionen können entweder direkt beim Starten des Compilers als Parameter übergeben werden, oder in ungeschachtelten Kommentaren gesetzt werden. Dabei werden sie mit einem Dollarzeichen ("$") eingeleitet. Direkt dahinter folgt der Name der Option. Es wird Groß- und Kleinschreibung unterschieden. Als letztes Zeichen folgt ein "+", "-" oder "=". "+" schaltet die Option ein, "-" schaltet sie aus. "=" kann nur bei stapelbaren Optionen verwendet werden. Es setzt sie zurück auf den Wert vor dem letzten "+" oder "-". Dadurch kann man die Optionen leicht schachteln.

Beispiel:

```
(* $RangeChk- hier keine Bereichskontrolle *)
    ....
(* $RangeChk= Bereichskontrolle wieder auf ursprünglichen Wert *)
```

Hier nun die Beschreibungen der einzelnen Optionen:

In Klammern jeweils der Gültigkeitsbereich, der voreingestellte Wert und evtl. der Compilerparameter, der diese Option ausschaltet. Stapelbare Optionen beziehen sich auf den gesamten Text. Prozedurbezogene Optionen gelten für diejenige Prozedur, deren Prozedurkörper als nächstes folgt.

CaseChk (stapelbar, +, c):

Wenn gesetzt, wird Code erzeugt, der prüft, ob ein Element in einer CASE-Anweisung ohne ELSE gefunden wurde, und anderenfalls das Programm abbricht.

CodeChip (bezogen auf Modul, -):

$CodeChip+ erzwingt, daß der Codeteil dieses Moduls in Chip-Memory (siehe [cbm:exec]) geladen wird. Dies kann nötig sein, wenn der Code z.B. Grafikdaten enthält.

CopyArrays (bezogen auf Prozedur, -):

$CopyArrays- verhindert, daß offene Feldparameter auf den Stack kopiert werden. Sie werden stattdessen wie VAR-Parameter behandelt. Die Felder dürfen dann von der Prozedur nicht verändert werden.

$CopyArrays- kann Prozeduren, die Strings als Parameter haben, optimieren. Konstante Zeichenketten können nicht an VAR-Paramter zugewiesen werden. Durch diese Option werden die offenen Feldparameter alle zu Pseudo-VAR-Parametern.

DataChip (bezogen auf Modul, -):

$DataChip+ erzwingt, daß der Konstantenbereich dieses Moduls in Chip-Memory (siehe [cbm:exec]) geladen wird. Dies ist z.B. dann nötig, wenn man eine Wellenform für Musik als eine Feldkonstante speichert.

DeallocPars (bezogen auf Prozedur, +):

$DeallocPars- verhindert, daß eine Prozedur ihre Paremeter vom Stack entfernt. Dadurch kann diese Prozedur von C-Routinen aus aufgerufen werden. C-Routinen geben ihre Parameter nicht selbst frei.

EntryExitCode (bezogen auf Prozedur, +):

$EntryExitCode- verhindert, daß Eingangs- und Ausgangscode für die folgende Prozedur erzeugt wird. Im Einganscode wird normalerweise der Speicher für die lokalen Variablen angelegt und die offenen Feldparameter werden kopiert. Im Ausgangscode wird der Speicher der lokalen Variablen und der Parameter freigegeben und an die Stelle des Prozeduraufrufs zurückgesprungen.

Bei INLINE-Maschinencode-Prozeduren oder Prozeduren, die nur Daten enthalten, ist es oft sinnvoll, diese Option zu löschen.

Implementation (bezogen auf Modul, +):

$Implementation- verhindert, daß für dieses Modul eine Objektdatei erzeugt wird. Dies ist in Modulen, die nur Typen und Konstanten enthalten, sinnvoll.

Ein solches Modul darf keine Variablen, Stringkonstanten und Prozeduren enthalten. auch keine strukturierte Konstanten.

NilChk (stapelbar, +, n):

Der Compiler erzeugt normalerweise Überprüfungscode, der abfängt, wenn ein Programm einen Zeiger, der NIL enthält, z.B. mit "^" dereferenziert. Diese Überprüfung macht Programme, in denen viel mit Zeigern gearbeitet wird, sehr sicher. Der Nachteil ist, daß diese Überprüfung Programme oft stark verlängert und verlangsamt.

$NilChk- schaltet diese Überprüfung aus. Dies sollte jedoch nur in sorgfältig ausgetesteten Modulen gemacht werden.

OvflChk (stapelbar, +, v):

$OvflChk- schaltet die Überlaufsüberprüfung ab. Man spricht von einem Überlauf, wenn das Ergebnis einer Berechnung so groß ist, daß es nicht mehr in den gewählten Typ paßt. Beispiel:

```
TYPE i: INTEGER;
BEGIN
  i := 1000;
  i := i*i;
END;
```

Das korrekte Ergebnis wäre 1 000 000, was größer als MAX(INTEGER) ist. Deshalb würde dieses Programm normalerweise wegen eines Überlaufs abbrechen. Die Überlaufskontrolle verlängert und verlangsamt Programme kaum, da für sie nur ein kurzer Befehl (TRAPV) hinter den Berechnungen eingefügt wird.

RangeChk (stapelbar, +, b):

$RangeChk- schaltet die Bereichskontrolle ab. Normalerweise wird beim Zugriff auf Feldelemente überprüft, ob der Index größer oder gleich 0 und kleiner als die Länge des Feldes ist, da sonst auf ungültige Daten zugegriffen würde. Entsprechend wird der Bereich bei SET-

Elementen und bei Umwandlung mit der Standardprozedur SHORT() überprüft.

ReturnChk (stapelbar, +, r):

Eine Prozedur, die ein Ergebnis liefert, muß mit einer RETURN-Anweisung beendet werden. Wurde das RETURN vergessen, wird dies normalerweise während der Laufzeit abgefangen. $ReturnChk- schaltet diese Überprüfung ab.

SaveAllRegs (bezogen auf Prozedur, -):

$SaveAllRegs+ bewirkt, daß die Prozedur alle Register des 68000'er Prozessors rettet, so daß sie nach einem Prozeduraufruf unverändert sind. Dies kann z.B. wichtig sein, wenn diese Prozedur von Assemblercode aus aufgerufen wird.

SaveRegs (bezogen auf Prozedur, -):

Ist die Option SaveRegs gesetzt, werden ähnlich wie bei SaveAllRegs die Register D2-D7 und A2-A7 unverändert zurückgegeben. Dies ist nötig, wenn die Prozedur z.B. von C-Routinen aus aufgerufen wird.

StackChk (stapelbar, +, s):

Lokale Variablen und Prozedurparameter werden normalerweise auf dem Stapelspeicher abgelegt. Dieser Speicher ist jedoch nicht unbegrenzt, sondern kann bei vielen geschachtelten Prozeduraufrufen und vielen lokalen Variablen schnell ausgehen. Normalerweise wird ein Oberon-Programm abgebrochen, wenn dieser Speicher ausgeht.

$StackChk- schaltet diese Überprüfung aus. Programme ohne Stackkontrolle können bei zu wenig Stapelspeicher Abstürze verursachen, auch wenn das Programm völlig korrekt ist. Daher sollte diese Option sehr vorsichtig verwendet werden.

Die Stackkontrolle muß in Prozeduren abgeschaltet werden, die als Interrupt verwendet werden sollen.

TypeChk (stapelbar, +, t):

Beim Zugriff auf erweiterte RECORDs in der WITH-Anweisung oder mit dem Typeguard (mit x(Typ).y), wird geprüft, ob der Wert auch vom Typ der Erweiterung ist. Ist dies nicht der Fall, wird ein Programm abgebrochen, da auf ungültige Daten zugegriffen würde.

$TypeChk- schaltet diese Überprüfung aus. Dies kann in Anweisungen wie der folgenden bedenkenlos gemacht werden:

```
IF p IS ABC THEN
  WITH p: ABC DO
    ...
  END;
END;
```

Hier ist sichergestellt, daß p vom Typ ABC ist. Sonst sollte die Typüberprüfung jedoch nur bei zeit- und speicherkritischen Programmen, die sorgfältig ausgetestet wurden, ausgeschaltet werden.

Die verschiedenen Speichermodelle:

Kurze Programme und Programme mit wenigen globalen Variablen können durch die Verwendung der kleinen Speichermodelle noch weiter optimiert werden. Dies muß dem Compiler und dem Linker jedoch durch das Setzen von Optionen mitgeteilt werden.

Das kleine Code-Modell:

Dieses Speichermodell wird durch die Compiler- und Linkeroption '-m' eingeschaltet. Dabei werden alle Aufrufe von Prozeduren aus importierten Modulen optimiert. Dies funktioniert jedoch nur, wenn der Code-Teil des Programms insgesamt höchstens 32KByte lang ist.

Ist er größer, erzeugt BLink zusätzlichen Code, damit das Programm dennoch gelinkt werden kann. Das ereugte Programm ist dann jedoch meist länger und langsamer als mit dem großen Code-Modell. Das gleiche passiert, wenn man beim Linken vergißt, die Option '-m' zu setzen.

Das kleine Daten-Modell:

Aktiviert wird dieses Speichermodell durch die Compiler- und Linkeroption '-d'. Dabei werden die Speicherbereiche der globalen Variablen aller Module zu einem Speicherblock zusammengefügt. Der Vorteil ist, daß beim Wechseln der globalen Variablen durch einen Aufruf einer Prozedur eines anderen Moduls, der Speicherblock nicht gewechselt werden muß. Damit fällt je Prozedur eine ganze Menge Code weg. Zudem kann schneller auf Variablen importierter Module zugegriffen werden.

Ein weiterer Vorteil dieses Speichermodells ist, daß der Speicher der globalen Variablen erst während der Laufzeit alloziert wird. Dadurch können diese Programme resident gehalten werden und mehrmals gestartet werden, es wird ja bei jedem Start ein neuer Satz globaler Variablen angelegt.

Die Einschränkung dieses Speichermodells wird von den meisten Programmen erfüllt: Es darf maximal 32KByte globale Variablen je Porgramm geben. Bei großen Feldern kann diese Grenze durch die Verwendung von Zeigern und das Allozieren des Speichers während der Laufzeit eingehalten werden.

Wird ein Programm mit dem kleinen Datenmodell compiliert, so müssen alle Module mit diesem Modell compiliert werden, und auch beim Linken muß die Option '-d' gesetzt sein. Bei der Compilation werden Objektdateien mit der Endung '.objs' erzeugt, um sie von den normal compilierten unterscheiden zu können.

Dadurch, daß Programme bei dieser Option reentrant sind, ergibt sich

noch eine Reihe von Vorsichtsmaßnahmen für die Programmierung, vor allem bei der Verwendung der 'low-Level' Fähigkeiten dieses Compilers:

- Assemblercode, der vom Oberon-Programm aufgerufen wird, darf selbst keine globalen Variablen verwenden.

- Bei der Programmierung von Exceptions, Interrupts etc. und beim Starten von neuen Tasks muß sichergestellt sein, daß die betroffene Prozedur entweder keine globalen Variablen verwendet oder daß das Register A5 korrekt geladen wird.

- In Oberon geschriebene Libraries und Devices müssen A5 in jeder Prozedur, die von außen aufgerufen werden kann, vor der Verwendung von globalen Variablen korrekt setzen.

- A5 darf während dem Programmlauf nicht verändert werden.

Ansonsten dürfte es jedoch keine Probleme geben, so daß diese Option bei den meisten Programmen gesetzt werden kann.

Kombination der Speichermodelle:

Es ist natürlich möglich, die beiden Speichermodelle zu kombinieren, um die Vorteile beider Modelle auszunutzen. Dies ist besonders bei der Programmierung kleiner CLI-Utilities sinnvoll, bei denen oft auch die Residentfähigkeit eine wichtige Rolle spielt.

9. Unterschiede zwischen Oberon und Modula-2:

Oberon ist eine Weiterentwicklung der Sprache Modula-2 und führt somit die lange Sprachentwicklung, die mit Algol begann und über Pascal zu Modula führte, fort. Bei der Entwicklung von Oberon wurde besonderer Wert darauf gelegt, die Sprache zu vereinfachen und gleichzeitig ihre Möglichkeiten zu erweitern. Dazu wurde sie um ein paar Fähigkeiten ergänzt, während einige Fähigkeiten, die entweder überflüssig oder gar hinderlich waren, gestrichen wurden. Das Ergebnis ist eine leicht erlernbare, einfache Sprache, die dennoch viele neue Möglichkeiten bietet und den bisherigen Sprachen in mancher Hinsicht überlegen ist.

Neue Fähigkeiten von Oberon:

Die wohl wichtigste Neuerung in Oberon ist die Erweiterbarkeit von RECORD-Typen. In Modula war es bereits möglich, auf der Basis von bestehenden Modulen und Prozeduren neue Prozeduren aufzubauen. Diese Erweiterbarkeit der Programme wird nun auch mit den Datentypen möglich. Durch diese Möglichkeit wird Oberon zu einer objektorientierten Sprache. Das Entwickeln von neuen Modulen auf der Basis einer großen Modulbibliothek wird erleichtert, da die erweiterten Typen weiterhin kompatibel zu ihren Basistypen bleiben, sodaß bereits bestehende Prozeduren auch mit den neuen Typen arbeiten können.

Ein schönes Beispiel für die Anwendungsmöglichkeiten, die die Recorderweiterungen bieten, sind die Typen der Elemente in einem Baum. Zunächst der Knoten, der kein erweitertes Record ist:

```
TYPE
  NodePtr = POINTER TO Node;
```

```
Node = RECORD
  left,right: NodePtr;
  key: INTEGER;
END;
```

Der Baum soll aus diesen Knoten aufgebaut und nach Node.key sortiert sein. Eine Prozedur, die einen Knoten des Baumes sucht, sieht wie folgt aus:

```
PROCEDURE Find(root: NodePtr; k: INTEGER): NodePtr;
BEGIN
  WHILE (root#NIL) AND (root.key#k) DO
    IF root.key<k THEN root := root.right
                  ELSE  root := root.left END;
  END;
  RETURN root;
END Find;
```

Entsprechend eine Prozedur zum Einfügen neuer Elemente in den Baum:

```
PROCEDURE Add(VAR root: NodePtr; el: NodePtr);
BEGIN
  IF root=NIL THEN root := el
  ELSIF root.key>el.key THEN Add(root.left,el)
                        ELSE  Add(root.right,el) END;
END Add;
```

Nun sind jedoch noch keine Daten in dem Baum gespeichert. Um dies zu tun, kann der Knotentyp erweitert werden. Nehmen wir an, wir schreiben ein CAD-Programm, das Kreise und Rechtecke zeichnet:

```
TYPE
  RectPtr = POINTER TO Rect;
```

```
Rect = RECORD (Node)
  x,y,b,h: INTEGER
END;

CirclePtr = POINTER TO Circle;
Circle = RECORD (Node)
  mx,my: INTGER;
  r: INTEGER;
END;
```

Diese 2 Records enthalten die Elemente left, right und key des Basistyps Node. Zudem enthalten sie die hier definierten zusätzlichen Elemente. Diese beiden Typen sind zuweisungskompatibel zum Typ Node. Entsprechendes gilt für die Zeiger auf diese Typen. Daher können sie einfach mit Add() in den Baum eingefügt werden. In dem gleichen Baum können sowohl Rechtecke als auch Kreise gespeichert werden. Es ist also ein sogenannter inhomogener Baum. Die Prozedur zur Eingabe eines Rechtecks kann folgendermaßen aussehen:

```
PROCEDURE DefineRect(VAR root: NodePtr; k: INTEGER);

VAR r: RectPtr;

BEGIN
  NEW(r); IF r=NIL THEN HALT(20) END;
  r.key :=  k;
  WaitMouseButton;
  r.x := MouseX;
  r.y := MouseY;
  WaitMouseButton;
  r.b := MouseX - r.x;
  r.h := MouseY - r.y;
  Add(root,r);
END DefineRect;
```

Wenn nun ein Element dieses Baumes mit einem bestimmten *key* gezeichnet werden soll, kann es mit der Prozedur Find() gesucht werden. Nun muß jedoch festgestellt werden, von welchen Typ dieses Element ist, d.h. ob es ein Rechteck, ein Kreis oder noch etwas anderes ist. Dies geschieht mit dem Typ-Test, der mit dem Schlüsselwort IS gemacht wird.

Danach muß dem Compiler noch mitgeteilt werden, daß das Element ab sofort nicht nur ein Knoten ist, sondern ein Rechteck oder ein Kreis. Dies kann in einem Ausdruck mit einem Typeguard geschehen, also z.B. mit node(Rect).x. Soll über ein größeres Programmstück ein anderer Typ angenommen werden, kann dies mit Hilfe der WITH-Anweisung geschehen.

Hier die Prozedur zum Zeichnen:

```
PROCEDURE Draw(root: NodePtr; k: INTEGER);

VAR node: NodePtr;

BEGIN
  node := Find(root,k);
  IF node#NIL THEN
    IF node IS Rect THEN
      WITH node: Rect DO
        DrawRect(node.x,node.y,node.b,node.h);
      END;
    ELSIF node IS Circle THEN
      DrawCircle(node(Circle).mx,
                 node(Circle).my,
                 node(Circle).r);
    END;
  END;
END Draw;
```

Man kann nun annehmen, die Typerweiterung könne auch in

Modula-2 leicht simuliert werden. Die Typen für Rechteck und Kreis könnten dann folgendermaßen definiert werden:

```
TYPE
  RectPtr = POINTER TO Rect;
  Rect = RECORD
    node: Node;
    x,y,b,h: INTEGER
  END;

  CirclePtr = POINTER TO Circle;
  Circle = REDORD
    node: Node;
    mx,my: INTGER;
    r: INTEGER;
  END;
```

In Modula-2 hat man dann etwas mehr Schreibarbeit beim Zugriff auf den Knoten, nämlich Rect.node.left statt Rect.left. Zudem sind die Typen nicht zuweisungskompatibel zu Node, es muß stattdessen Rect.node oder Circle.node geschrieben werden. Dies ist jedoch alles nicht der entscheidende Nachteil von Modula. Das Problem, das man in Modula hat, ist eine saubere Überprüfung des Typs, wenn man lediglich einen Zeiger auf einen Knoten hat, wie dies in der Prozedur Draw() der Fall ist.

In diesem Beispiel ist die Typüberprüfung überhaupt nicht möglich. Aber selbst wenn man 'von Hand' noch ein Feld einfügt, das Informationen über den Typ enthält, kommt man um eine unsaubere Typumwandlung nicht herum.

Eine andere Möglichkeit, wie man diese Typen in Modula-2 realisieren könnte, wären variante Records. Dabei geht jedoch die Fähigkeit der beliebigen Erweiterbarkeit verloren, es muß von vorneherein bekannt sein, welche zusätzlichen Informationen in den verschiedenen Records enthalten sein müssen und welche Erweiterungen nötig sind.

Bei der Definition von exportierten Recordtypen müssen in Oberon alle nach außen sichtbaren Elemente mit einen Sternchen markiert werden. Dies erweitert die Möglichkeiten der opaken Datentypen, die es in Modula-2 gibt. In Oberon kann für jedes Element einzeln entschieden werden, ob es öffentlich ist oder nur modulintern verwendet werden kann. Ein opaker Typ kann einfach dadurch erzielt werden, daß in einer Recordtypdeklaration kein Recordelement nach außen sichtbar ist. Auf den Inhalt eines solchen Typs kann man in anderen Modulen zwar zunächst nicht zugreifen, man hat aber weiterhin die Möglichkeit, diese Records zu erweitern und kann somit eigene Daten darin speichern.

Eine weitere neue Möglichkeit, die Oberon bietet, ist das Konzept des Typeinschlusses. Es gibt eine Hirarchie von numerischen Typen, wobei die oberen Typen die jeweils tieferliegenden einschließen. Zuweisungen der unteren Typen an die oberen sind möglich. Die Zuweisungsmöglichkeiten sind hier durch Pfeile markiert:

SHORTINT -> INTEGER -> LONGINT -> REAL -> LONGREAL

Wird in Ausdrücken mit verschiedenen Typen gerechnet, ist das Ergebnis immer von dem Typ, der die Typen beider Operanden einschließt. Beispiel:

```
VAR
  i : INTEGER;
  j : LONGINT;
  k : REAL;
  l : LONGREAL;
```

Ausdruck	Typ
k*i	REAL
i*j+l	LONGREAL
j/i	REAL
j DIV i	LONGINT

Das Ergebnis einer Division mit "/" ist immer ein Real-Typ, auch wenn die Operanden nur von Integertypen sind.

Mit den Variablen von oben sind z.B. folgende Zuweisungen möglich:

```
j := i;
l := j;
l := k+10;
```

Zuweisungen wie i := j oder j := k sind jedoch nicht erlaubt.

Prozedurtypen werden in Oberon ähnlich definiert wie Prozeduren selbst. Dabei werden dann dummy-Parameterbezeichner benutzt, die als Kommentar dienen können. Beispiel:

```
TYPE MyProc = PROCEDURE(i,j: INTEGER);
```

Dieser Prozedurtyp wäre in Modula-2 PROCEDURE(INTEGER, INTEGER).

Fähigkeiten, die gegenüber Modula-2 fehlen:

Typen:

Variante Records und Opake Typen gibt es in Oberon nicht mehr. Sie können beide durch die neuen Möglichkeiten, die sich durch die Erweiterbarkeit von Records ergeben, ersetzt werden. Durch die Möglichkeit, bei Recordtypdeklarationen die Sichtbarkeit jedes einzelnen Elementes festzulegen, bietet Oberon eine deutlich flexiblere Methode zum 'Information-Hiding'.

Aufzählungstypen gibt es in Oberon nicht mehr, da sie leicht durch Konstanten ersetzt werden können und über Modulgrenzen hinweg nicht erweiterbar sind. So kann z.B. der Aufzählungstyp

```
TYPE Ampel = (rot,gelb,gruen);
```

durch die 3 Konstanten

```
CONST rot = 0; gelb = 1; gruen = 2;
```

ersetzt werden. Statt des Typs Ampel kann dann SHORTINT verwendet werden. Möchte man noch weitere Elemente hinzufügen, wie dies z.B. bei einer Umweltampel nötig ist, ist dies jetzt leicht möglich:

```
CONST blau = 3;
```

Oberon kennt keine Unterbereichstypen mehr. In Oberon muß man stattdessen deren Basistyp verwenden.

Durch das Fehlen der Aufzählungs- und Unterbereichstypen verliert die Möglichkeit der Definition neuer SET-Typen ihren Sinn. Stattdessen gibt es nur noch einen Standard-SET-Typ (Diese Oberon-Implementation kennt zusätzlich die Typen SHORTSET und LONGSET).

In Oberon gibt es die Typen CARDINAL und LONGCARD nicht mehr. Es gibt kaum Programme, die den vollen CARDINAL-Zahlenbereich ausgenutzt haben. Stattdessen kann also auch INTEGER verwendet werden. Durch das Fehlen der CARDINALs hat die Inkompatibilität zwischen INTEGER und CARDINAL ein Ende.

Zeiger in Oberon dürfen nur noch auf RECORDs und auf ARRAYs zeigen.

Der Indextyp von ARRAYs ist in Oberon immer Integer. Bei der Definition von ARRAY-Typen werden keine 2 Grenzen, sondern nur noch die Länge des Feldes angegeben. Die Indizes reichen dann von 0 bis zur Länge minus 1.

Module:

Lokale Module gibt es in Oberon nicht mehr, da sie in Modula-2 kaum benutzt wurden.

Importierte Bezeichner können nicht mehr mit "FROM Module IMPORT x" direkt importiert werden, sondern müssen qualifiziert importiert werden. Um dies zu vereinfachen, kann der Modulname in der Importliste abgekürzt werden, also "IMPORT m: Module;" und später im Modul "... m.x ...".

Durch den qualifizierten Import werden Module leichter verständlich, da die Herkunft aller Bezeichner sichtbar ist.

In Modula gab es drei verschiedene Arten von Modulen, nämlich Definitionen, Implementationen und Hauptmodule. In Oberon gibt es nur noch eine Art von Modul, das die Funktion aller drei Modultypen übernimmt. Die exportierten Bezeichner, die in Modula in der Definition aufgelistet wurden, werden durch Sternchen markiert. Hauptmodule werden nicht besonders gekennzeichnet. Alle Module können als Hauptmodul dienen und zu ausführbaren Programmen gelinkt werden.

Anweisungen:

Die WITH-Anweisung hat in Oberon eine völlig andere Aufgabe als in Modula-2 (siehe oben). Wie bei importierten Bezeichnern müssen die Recordelemente qualifiziert angewählt werden.

In Oberon gibt es keine FOR-Schleifen mehr. Diese wurden ohnehin selten verwendet und können leicht durch andere Schleifen ersetzt werden.

Niedere Funktionen:

Die in Modula-2 möglichen Typkonvertierungen wurden in Oberon

fallengelassen. Das Modul SYSTEM, das viele Programmierer ständig importiert haben, enthält keine Datentypen mehr, sondern nur noch niedere Prozeduren.

Module, die SYSTEM importieren gelten als nicht portierbar. SYSTEM soll in gewöhnlichen Modulen nicht verwendet werden, da es dort immer vermieden werden kann.

10. Schnittstellen zu Libraries und Assemblercode:

Definition von absoluten Variablen:

Unter absoluten Variablen versteht man Variablen, die an einer festen Speicheradressen stehen. Davon gibt es im Amiga eigentlich nur eine einzige: Execbase, die Basisadresse der Exec-Library. Sie steht an Adresse 4.

Zusätzlich können absolute Variablen verwendet werden, um die Hardware-Register direkt anzusprechen. Dies ist in vielen Fällen sehr praktisch, meist jedoch auch sehr schlechter Stil. Wer sauber programmiert, sollte über den Umweg Devices auf die Hardware zugreifen.

Absolute Variablen werden ähnlich wie normale Variablen definiert. Die Adresse ist eine Konstante vom Typ LONGINT. Sie wird in eckige Klammern gesetzt und hinter den Variablenbezeichner (bei exportierten Variablen hinter das Sternchen) geschrieben.

Beispiel aus Exec:

```
VAR
  exec* [4] : ExecBasePtr;
```

Deklaration und Aufruf von Library-Prozeduren:

Um auf die Routinen einer Amiga-Library zugreifen zu können, muß zunächst eine globale Variable deklariert werden, die die Basisadresse der Library enthält, also den Wert, den Exec.OpenLibrary() zurückgibt. Nun müssen zu allen Prozeduren die sogenannten Offsets bekannt sein. Sie können der Literatur zum Amiga-Betriebssystem entnommen werden.

Die Basisadresse einer Library muß eine globale Variable sein, die im selben Modul, in dem sich auch die Library-Prozedurdeklarationen befinden, definiert wird.

Die Deklaration der Prozeduren besteht aus einem normalen Prozedurkopf, wie bei allen anderen Oberonprozeduren auch. Dabei wird hinter den Prozedurbezeichner in geschweiften Klammern der Name der Basisvariablen und hinter einem Komma der Libraryoffset geschrieben. Da die Parameter einer Libraryprozedur gewöhnlich in Registern übergeben werden, müssen die Nummern der verwendeten Register in geschweifte Klammern hinter die jeweiligen Parameternamen geschrieben werden.

Beispiel:

```
PROCEDURE TextLength*{gfx,- 54}(rp{9}:RastPortPtr;
                                string{8}:e.ADDRESS;
                                count{0}:LONGINT):INTEGER;
```

Diese Prozedur hat den Offset -54. Der Zeiger auf die Library-Struktur befindet sich in der Variablen gfx. Es werden 3 Parameter in den Registern A1, A0 und D0 übergeben. Das Ergebnis der Prozedur wird immer in D0 zurückgegeben. Hier ist es ein INTEGER-Wert.

Bei VAR-Registerparametern wird nicht der Wert selbst, sondern ein Zeiger auf den übergebenen Wert in das Register geladen.

So deklarierte Libraryprozeduren können genauso wie alle anderen Oberonprozeduren aufgerufen werden, ohne daß man etwas Besonderes beachten muß. Dies kann auch von anderen Modulen aus geschehen. Es wird dabei direkt Code zum Aufrufen der Prozedur erzeugt, und nicht der Umweg über eine zweite Prozedur gegangen.

Wenn Libraryprozeduren aufgerufen werden, muß sichergestellt sein, daß die Library auch vorher geöffnet wurde. Die Amiga-Interface-

Module zu diesem Compiler, die den Zugriff auf die Libraries ermöglichen, öffnen sie alle selbständig.

Zugriff auf Variablen und Prozeduren von Assemblerroutinen

Dieser Compiler erlaubt es, in Assembler geschriebene Routinen aufzurufen und auf Variablen im Assemblercode zuzugreifen. Das Assemblerprogramm muß dabei nicht mit INLINE() in den Text eingefügt werden, sondern kann mit einem normalen Assembler, der Amiga-Objektdateien erzeugt, assembliert werden. Die billigste und eine der besten Lösungen ist der Assembler "a68k", der auf Public Domain Disketten erhältlich ist.

Um Bezeichner (Labels) vom Assemblercode aus zu exportieren, müssen sie mit "XDEF <label>" gekennzeichnet werden. Diese Bezeichner werden dann in die Objektdatei geschrieben. Um von Oberon aus darauf zuzugreifen, muß dem Compiler noch mitgeteilt werden, wie diese Bezeichner heißen und von welchem Typ sie sind. Bei Assemblerroutinen geschieht dies durch eine Prozedurdeklaration ähnlich wie bei Library-Prozeduren. In die geschweiften Klammern hinter den Prozedurbezeichner kommt hier jedoch ein konstanter String, der dem Namen des Labels entspricht.

Auf Variablen wird ähnlich wie bei absoluten Variablen zugegriffen. In die eckigen Klammern hinter den Variablenbezeichner kommt hier auch ein konstanter String für den Labelnamen.

Beispiel:

Folgende Assemblerroutine berechnet den größten gemeinsamen Teiler der beiden LONGINTs, die in D0 und D1 übergeben werden. Das Ergebnis wird in D0 zurückgegeben. In der INTEGER-Zahl "cnt" werden die benötigten Schleifendurchläufe gezählt.

```
        XDEF    cnt
        XDEF    GGT

        SECTION     "GGT",BSS

cnt:    dc.w    0

        SECTION     "GGT",CODE

GGT:    clr     cnt
loop:   addq    #1,cnt
        sub.l   D0,D1
        beq.s   done
        bpl.s   loop
        neg.l   D1
        sub.l   D1,D0
        bra.s   loop
done:   rts

        END
```

Diese Routine kann nun z.B. mit "a68k GGT.asm" assembliert werden. Das Ergebnis ist die Objektdatei GGT.o.

Im Oberon-Programm können die Variable cnt und die Routine GGT z.B. wie folgt verwendet werden:

```
MODULE Demo;

IMPORT io;

VAR
  zaehler["cnt"]: INTEGER;
  a,b: LONGINT;

PROCEDURE ggt{"GGT"}(a{0},b{1}: LONGINT): INTEGER;
```

```
BEGIN
  REPEAT
    io.WriteString("a> ")
  UNTIL io.ReadInt(a) AND (a>0);
  REPEAT
    io.WriteString("b> ")
  UNTIL io.ReadInt(b) AND (b>0);
  io.WriteString("GGT: "); io.WriteInt(ggt(a,b),8); io.WriteLn;
  io.WriteString("cnt: "); io.WriteInt(zaehler, 8); io.WriteLn;
END Demo.
```

Beim Linken müssen die Objektdateien der Assemblerteile extra angegeben werden, da OLink nur Oberon-Objektdateien finden kann. Dies geschiet durch einen weiteren Parameter an OLink, der hinter 'OBJ' angegeben wird. So kann das Programm von oben folgerndermaßen gelinkt werden:

```
OLink Demo OBJ GGT.o
```

Eine andere Möglichkeit, die Assemblerroutinen automatisch einbinden zu lassen, besteht darin, sie mit der Objektdatei eines Oberon-Moduls mit Hilfe des CLI-Befehls 'JOIN' zu verbinden. Bei dem Beispiel von oben kann man z.B. folgendes Modul schreiben:

```
MODULE GGT;

VAR cnt*["cnt"]: INTEGER;

PROCEDURE GGT*{"GGT"}(a{0},b{1}: LONGINT): INTEGER;

END GGT.
```

Dieses Modul wird nun zunächst ganz normal compiliert. Danach gibt man folgende CLI-Befehle ein:

```
JOIN GGT.obj GGT.o AS T:GGT.obj
COPY T:GGT.obj TO GGT.obj
```

Danach enthält die Objektdatei GGT.obj auch die Assemblerroutine. Alle Module, die GGT oder cnt aus diesem Modul importieren, können nun auch einfach mit OLink gelinkt werden. Diese Methode funktioniert auch dann, wenn in dem Oberonmodul sowohl Oberon als auch Assemblerprozeduren definiert werden.

Manchmal ist der umgekehrte Weg nötig, nämlich daß von Assemblercode aus Oberonprozeduren aufgerufen werden sollen. Dies geschieht mit dem Pseudo-Opcode "XREF" im Assemblertext. Dabei muß hinter XREF der Name der Prozedur angegeben werden, die aufgerufen werden soll. Dieser Name ist immer qualifiziert, das heißt er besteht aus dem Modulnamen, einem Punkt und dem Prozedurnamen. Folgendes Beispielprogramm gibt mit Hilfe von io.WriteString eine Zeichenkette auf dem Bildschirm aus:

```
        XREF    io.WriteString
        XDEF    demo

demo:   pea     hello(PC)           ; Stringadresse
        move    #14,-(A7)           ; Stringlänge
        jsr     io.WriteString      ; Prozedur aufrufen
        rts                         ; und zurück
hello:  dc.b    "Hello World!",10,0

        END
```

Ein Modul, das diese Prozedur aufruf, sieht z.B. folgendermaßen aus:

```
MODULE Demo;

IMPORT io;

PROCEDURE demo{"demo"};
```

```
BEGIN
  demo;
END Demo.
```

Dabei ist wichtig, daß das Modul io vom Oberonmodul importiert wird, da die Assemblerroutine die Prozeduren aus io nur dann verwenden darf, wenn io vorher korrekt geöffnet wurde. Oberonprozeduren sollten von Assembler aus nur dann aufgerufen werden, wenn man genau weiß, wie die Parameterübergabe von diesem Compiler realisiert wird.

Für das Linken gilt hier das gleiche wie oben bei reinen Assemblerprozeduren.

Schwieriger ist es, von Assembler aus auf Variablen des Oberon-Prorgramms zuzugreifen. Man sollte dann lieber die Variablen als Parameter an die Assemblerroutine übergeben. Alle globalen Variablen stehen hinter dem Label "<Modulname>_vars". Man könnte nun direkt über dieses Label auf die Variablen zugreifen, wenn man sich die Offsets 'von Hand' ausrechnet. Dies ist jedoch gefährlich, da die Adressen der Variablen bei künftigen Compilerversionen anders sein können. Besser ist es, alle für die Assemblerroutine wichtigen Variablen in einer Struktur (mit STRUCT) zusammenzufassen, und diese Struktur der Routine als VAR-Parameter, am besten in einem Adressregister, zu übergeben.

11. Wie erzeuge ich schnellen Code?

Dieser Compiler erzeugt im allgemeinen kurzen und schnellen Code. Darauf wurde auch während der gesamten Entwicklungszeit besonderer Wert gelegt. Dennoch kann man, wenn man über ein paar Eigenheiten des Compilers und des erzeugten Codes Bescheid weiß, noch schnellere Programme schreiben, ohne auf den umständlichen Assembler umsteigen zu müssen.

Die in diesem Kapitel vorgestellten Möglichkeiten, den erzeugten Code vom Quelltext aus zu verbessern sollten, jedoch nicht angewendet werden, wenn dadurch Programmteile, die nicht besonders zeit- und speicherplatzkritisch sind, unübersichtlicher und dadurch schwerer zu pflegen werden.

Speichermodelle

Die einfachste Möglichkeit, den Code zu verbessern, ist die Verwendung der kleinen Speichermodelle, wo dies möglich ist. Dabei kann das kleine Datenmodell auch dann meistens verwendet werden, wenn man mit großen Feldern arbeiten möchte. Dazu deklarariert man statt den Feldvariablen Zeigervariablen auf den Feldtyp und alloziert den Speicher dafür während der Laufzeit.

Variablen:

Einer der wichtigsten Punkte bei der Geschwindigkeit ist wohl die Dauer eines Zugriffs auf eine Variable. Auf Variablen wird normalerweise (beim kleinen Datenmodell immer) relativ zu einem Adreßregister, mit einem 16-Bit breiten Offset, zugegriffen. Dies ist deutlich effektiver als der Weg über absolute Adressen. Der Nachteil ist, daß bei einem globalen Variablenbereich eines Moduls, der größer als 32 KByte ist, bei den zuletzt definierten Variablen dennoch die absolute Adressierung verwendet werden muß. Daher sollte der Variablenbereich je Modul immer möglichst kleiner als 32K gehalten werden. Bei

größeren Variablenbereichen sollten die oft verwendeten, kleineren Variablen vor den seltener verwendeten deklariert werden. Die Größe des Variablenbereichs wird nach der Compilation hinter dem Wort "BSS:" ausgegeben.

Eine Sonderstellung nimmt die jeweils zuerst definierte globale oder lokale Variable ein. Werden mehrere Variablen in der gleichen Zeile definiert, betrifft dies diejenige, die ganz rechts, direkt vor dem Doppelpunkt steht. Auf sie kann indirekt über ein Adreßregister ohne einen zusätzlichen Offset zugegriffen werden. Dadurch wird der Code bei jedem Zugriff 2 Bytes kürzer und schneller. Man sollte also Variablen, die besonders oft verwendet werden, zuerst definieren.

Wird auf Variablen aus importierten Modulen zugegriffen, so wird die absolute Adressierung verwendet. Bei sehr vielen Zugriffen kann es sich daher lohnen, externe Variablen in Variablen des Moduls, in dem sie oft verwendet werden, zu kopieren. Dabei muß man jedoch darauf achten, daß die Variablen zurückgeschrieben werden, wenn sie verändert wurden. Beim kleinen Datenmodell kann auf importierte Variablen gleich schnell zugegriffen werden wie auf die lokalen Variablen.

Bei geschachtelten Prozeduren wird der Zugriff auf lokale Variablen umschließender Prozeduren sehr kompliziert, zeit- und speicheraufwendig. Um dies zu vermeiden, können oft verwendete Variablen aus äußeren Prozeduren in globale oder zur inneren Prozedur lokale Variablen kopiert werden.

Strukturanweisungen:

Die REPEAT-Schleife wird etwas effizienter übersetzt als die Schleifen mit LOOP und WHILE. Falls es nicht zu Problemen führen kann, sollte man REPEAT verwenden.

Boolesche Ausdrücke und äquivalente geschachtelte IF-Anweisungen erzeugen bei der Compilation normalerweise den gleichen Code.

Beispiel:

Die Zeile

```
IF A THEN IF a() THEN b END END;
```

erzeugt exakt den gleichen Code wie

```
IF A & a() THEN b END;
```

Es sollte also immer die übersichtlichste und einleuchtendste Schreibweise verwendet werden.

Libraryaufrufe:

Vor dem Aufruf einer Amiga-Libraryprozedur muß die Basisadresse der Library in das Register A6 geladen werden. Bei mehrmaligen Aufrufen von Prozeduren derselben Library wird A6 nur einmal geladen, solange es nicht durch Aufrufe von anderen Prozeduren oder durch Strukturanweisungen wie Schleifen zerstört wurde. Um effizient mehrere Routinen derselben Library aufzurufen, sollten die Aufrufe daher möglichst direkt hintereinander geschrieben werden.

Typen:

Beim Arbeiten mit Integerzahlen sollte der Typ LONGINT nur dort verwendet werden, wo der Zahlenbereich von INTEGERs überschritten wird. Operationen mit LONGINT-Zahlen sind immer langsamer als solche mit INTEGERs, dies macht sich besonders bei Multiplikationen und Divisionen bemerkbar.

Werden SHORTINTs statt INTEGERs verwendet, verkürzen sich Code und Rechenzeit jedoch nicht. In manchen Fällen, wie zum Beispiel bei Multiplikationen und Divisionen, wird das Programm sogar länger und langsamer. SHORTINTs bieten nur den Vorteil, daß sie im Variablenbereich 1 Byte weniger benötigen.

Für ganze Zahlen sollte also so oft wie möglich der Typ INTEGER verwendet werden.

Bei der Definition von Feldern, auf die oft zugegriffen werden soll, ist es von großem Vorteil, wenn die Größe des Grundtyps des Feldes eine Potenz von zwei ist. Sonst muß bei jedem Zugriff auf ein Feldelement eine zeitraubende Multiplikation ausgeführt werden.

12. Der erzeugte Code:

In diesem Kapitel wird der interne Aufbau von Oberon-Programmen beschrieben. Informationen darüber dient vor allem Assemblerprogrammierern, die Einblicke in die innere Struktur der Programme benötigen. Die Informationen über den erzeugten Überprüfungscode können jedoch auch für reine Oberonprogrammierer von Nutzen sein.

Zum besseren Verständnis des Aufbaus ist die Kenntnis des Objektdateiformats nützlich. Eine relativ gute Beschreibung dieses Formats findet man in [rb:agb].

Aufbau von Oberonprogrammen:

Jedes Oberonmodul, außer das Modul OberonLib, kann zu einem ausführbaren Programm gelinkt werden. Daher enthalten alle Objektdateien als ersten Hunk (Objektdateien sind in bestimmte Unterblöcke, sog. Hunks unterteilt) Code, der die Open- und Close-Teile des Moduls aufruft. Dieser Hunk wird beim Linken bei allen importierten Modulen wegoptimiert.

Die Open- und Close-Segmente werden mit den Labels "<Modulname>_Vxxxxxxxxxxxx_open" bzw. "<Modulname>_close" eingeleitet. Dabei enthält das erste Label die Versionsnummer als Hexadezimalzahl.

Die globalen Prozeduren eines Moduls werden, zusammen mit ihren lokalen Prozeduren, in unterschiedliche Hunks gespeichert. Dadurch wird das optimierende Linken möglich. Alle exportierten Prozeduren erhalten ein Label der Form "<Modulname>.<Prozedurname>".

Die globalen Variablen eines Moduls stehen immer hinter dem Label "<Modulname>_vars". Dieses Label existiert auch dann, wenn im Modul keine globalen Variablen deklariert werden.

Entsprechend gibt es ein Label "<Modulname>_data", ab dem die Stringkonstanten stehen. Dieses Label fehlt jedoch, wenn in einem Modul keine konstanten Zeichenketten vorkommen.

Aufruf des Open- und Close-Codes:

Die letzte (INTEGER-) Variable im Variablenbereich jedes Moduls ist ein Zähler, der hochgezählt wird, wenn der Open-Code eines Moduls aufgerufen wird, und entsprechend bei der Ausführung des Close-Codes verringert wird. Beim Aufruf des Open-Teils wird geprüft, ob dieser Zähler nach dem Erhöhen den Wert 1 enthält. Ist dies nicht der Fall, wurde das Modul bereits geöffnet und es wird einfach zurückgesprungen. Ansonsten werden alle importierten Module geöffnet und dann der eigentliche Code zum Öffnen ausgeführt. So ist sichergestellt, daß jedes Modul, das in der Modulhierarchie niedriger gestellt ist, vor den jeweils höheren Modulen geöffnet wird.

Beim Schließen der Module vollzieht sich die gesamte Prozedur rückwärts: Die in der Hierarchie höheren Prozeduren müssen zuerst und die niedrigeren zum Schluß geschlossen werden. Dabei muß sichergestellt sein, daß kein Modul geschlossen wird, bevor ein Modul geschlossen wird, das dieses importiert. Dies wird dadurch erreicht, daß der Zähler, der beim Öffnen erhöht wurde, nun bei jedem Aufruf des Close-Teils verringert wird. Erst wenn er 0 ist, wird das Modul wirklich geschlossen und es werden die Schließprozeduren der importierten Module aufgerufen. Der Zähler ist am Schluß wieder 0, also der Anfangswert. Wenn ein Oberonprogramm resident geladen ist, kann es daher nun wieder gestartet werden, auch wenn es nicht mit dem kleinen Datenmodell compiliert wurde.

Zugriff auf Variablen:

Auf globale Variablen wird normalerweise über das Register A5 zugegriffen. A5 enthält die Basisadresse des Bereichs der globalen Variablen, auf die mit einem positiven Offset zugegriffen werden

kann. Ist der Variablenbereich größer als 32KByte, wird auf die 'hinteren' Variablen mit der absoluten Adressierung zugegriffen.

Die lokalen Variablen und Prozedurparameter werden auf dem Stapelspeicher abgelegt. Auf sie wird normalerweise relativ zum Register A7 zugegriffen. Dabei ist darauf zu achten, daß der Inhalt von A7 veränderlich ist und z.B. beim Aufruf von anderen Prozeduren verringert wird, wodurch sich der Offset der Variablen vergrößert.

Hat eine Prozedur offene Felder als Wertparameter, so müssen diese anfangs auf den Stack kopiert werden. Um auch dann auf die Variablen zugreifen zu können, werden ihre Basisadressen, die sonst in A7 stehen, nach A4 kopiert.

Wenn lokale Prozeduren auf die Variablen von umschließenden Prozeduren zugreifen müssen, verwenden sie einen Zeiger auf die Variablen der umschließenden Prozedur, der automatisch als interner Prozedurparameter übergeben wird. Bei tieferen Verschachtelungen muß man über mehrere dieser Zeiger auf die Variablen tiefergelegener Prozeduren zugreifen.

Auf Variablen externer Module wird (außer beim kleinen Datenmodell) absolut zugegriffen.

Zugriff auf String-, Feld - und Recordkonstanten:

Auf Konstanten wird immer mit absoluter Adressierung zugegriffen. Konstanten sind so 'selten', daß es sich nicht lohnen würde, für ihre Basisadresse ein Adreßregister zu verschwenden.

Der Aufbau des Stapelspeichers bei globalen Prozeduren:

In einer globalen Prozedur enthält der Stack folgende Daten:

Belegt von früher aufgerufenen Prozeduren	
Prozedurparameter	
Rücksprungadresse	
evtl. gerettetes A5	
Bereich der lokalen Variablen	
frei	< hierher zeigt A7

Die Prozedurparameter werden in der Reihenfolge, in der sie beim Prozeduraufruf geschrieben werden, auf den Stapel gelegt. Die Rücksprungadresse wird von dem JSR-Befehl auf den Stapel gelegt. Die aufgerufene Prozedur rettet, wenn dies nötig ist, zunächst A5, und lädt die Adresse der globalen Variablen dieses Moduls. Die lokalen Variablen stehen in der Reihenfolge ihrer Deklaration im Speicher, das heißt die erste Variable bei 0(A7) etc.

Der Aufbau des Stapelspeichers bei lokalen Prozeduren:

Belegt von früher aufgerufenen Prozeduren	
Prozedurparameter	
Zeiger auf lokale Variablen der umschließendender Prozedur	
Rücksprungadresse	
Bereich der lokalen Variablen	
frei	< hierher zeigt A7

Lokale Prozeduren können nicht aus anderen Modulen aufgerufen werden, weshalb A5 nicht gerettet werden muß. Stattdessen wird ein Zeiger auf den Variablenbereich der direkt umschließenden Prozedur übergeben. Dieser Zeiger steht beim Abarbeiten dieser Prozedur gewöhnlich in A7.

Format der offenen Feldparameter:

	< Adr. + 4+2*n
Zeiger auf Feldinhalt	
	< Adr. + 2*n
Länge in Dimension n	
	< Adr. + 2*(n-1)
etc.	
	< Adr. + 4
Länge in Dimension 1	
	< Adr. + 2
Länge in Dimension 0	
	< Adr.

Hat eine Prozedur offene Felder als Parameter, wird beim Aufruf zunächst die Adresse des Feldes und danach die Längen in den einzelnen Dimensionen, angefangen bei der höchsten Dimension, auf den Stack gelegt. Dabei sind die Längen INTEGER-Werte, sie dürfen also höchsten 7FFFH sein.

Beim Aufruf wird kein Unterschied zwischen Variablen- und Wertparametern gemacht. Wertparameter werden erst nach dem Aufruf der Prozedur auf den Stack kopiert.

Der erzeugte Überprüfungscode:

Stackkontrolle:

Zur Stackkontrolle wird am Anfang jeder Anweisungsfolge, die durch BEGIN oder CLOSE eingeleitet wird, der in dieser Anweisungsfolge durch lokale Variablen und andere Prozeduraufrufe maximal benötigte Stapelbereich mit OberonLib.StackChk geprüft. Dazu wird die Größe des Bereichs in D0 geschrieben und die Prozedur angesprungen.

Beispiel:

Eine leere exportierte Prozedur mit 12 Bytes lokale Variablen wird folgendermaßen übersetzt:

```
00000000 :  MOVE.L      A5,-(A7)
00000002 :  MOVEA.L     L000004(PC),A5
00000006 :  MOVEQ       #12,D0
00000008 :  JSR         $00000000
0000000E :  LEA         -12(A7),A7
00000012 :  LEA         12(A7),A7
00000016 :  MOVEA.L     (A7)+,A5
00000018 :  RTS
0000001A :  NOP
HUNK_EXT
relocatable definition: test_AC = 00000000  (0)
16 bit references on: test_AB
000000: 00000004 ....
32 bit references on: OberonLib.StackChk
000000: 0000000A ....
HUNK_END
```

Überlaufskontrolle:

Ein Überlauf wird bei Rechnungen an einem gesetzten Overflowflag erkannt. Um dieses zu erkennen, wird der TRAPV-Befehl verwendet.

Beispiel:

Die Anweisung INC(i) in Oberon wird folgendermaßen übersetzt:

```
0000001E :  ADDQ.W      #1,(A5)
00000020 :  TRAPV
```

Bereichskontrolle:

Die Bereichskontrolle überprüft beim Zugriff auf Feldelemente, bei Umwandlungen von Integerzahlen mit der Standardprozedur SHORT() und beim Arbeiten mit Settypen, ob der zulässige Wertebereich eingehalten wird.

Beispiel:

```
VAR s: ARRAY 80 OF CHAR;
     i: INTEGER;
     c: CHAR;

BEGIN
  c := s[i];
END
```

Für diese Zuweisung wird folgender Code erzeugt:

```
00000024 :  MOVE.W     80(A5),D7
00000028 :  CHK        #$004F,D7
0000002C :  MOVE.B     0(A5,D7.W),82(A5)
```

Kann der Bereich nicht mit CHK geprüft werden, wird stattdessen ein TRAP #0 verwendet.

Caseindex-kontrolle:

Diese Überprüfungsart verlangsamt Programme nicht und erzeugt lediglich einen TRAP #1-Befehl hinter allen CASE-Anweisungen, die keinen ELSE-Teil haben.

Return-Kontrolle:

Auch diese Überpüfung verlangsamt Programme nicht. Es wird lediglich bei allen Prozeduren, die ein Ergebnis zurückliefern, als letzter Befehl ein TRAP #4 erzeugt.

NIL-Zeiger Überprüfung:

Die Überprüfung, ob Zeiger, die dereferenziert werden, den Wert NIL enthalten ist relativ zu den anderen Überprüfungsarten sehr zeit- und speicheraufwendig.

Beispiel:

```
VAR i: INTEGER;
   p: POINTER TO RECORD a,b,c: INTEGER END;

BEGIN
  i := p.b;
END;
```

Für die Zuweisung wird folgender Code erzeugt:

```
00000020 :  MOVEA.L    2(A5),A4
00000024 :  MOVE.L     A4,D7
00000026 :  BNE.S      L00002A
00000028 :  TRAP       #3
0000002A :  MOVE.W     2(A4),(A5)
```

Hier mußten also ganze 3 Befehle eingefügt werden.

Typüberprüfung:

Wird auf Daten von erweiterten Records mit einem Typeguard oder mit der WITH-Anweisung zugegriffen, erzeugt der Compiler Code, der den Typ der Daten prüft. In jedem Record enthält das letzte (versteckte) Element Informationen darüber, ob das Record erweitert wurde oder nicht. In letzterem Fall enthält es den Wert 0.

Die Typüberprüfung erzeugt auch sehr viel Code. Dies wird besonders stark bemerkbar, wenn über mehrere Erweiterungen getestet wird.

Beispiel:

```
TYPE
  A = RECORD a,b: SHORTINT END;
  B = RECORD (A) c,d: INTEGER  END;
  C = RECORD (B) e,f: LONGINT  END;

VAR p,q: POINTER TO A;

BEGIN
 p(C).e := q(B).d;
END;
```

Diese Zuweisung erzeugt folgenden Code, wenn bei der Compilation die NIL-Kontrolle abgeschaltet war:

```
00000024 :  MOVEA.L    4(A5),A4
00000028 :  MOVE.L     A4,D7
0000002A :  BEQ.S      L000042
0000002C :  CMPI.L     #$75000001,2(A4)
00000034 :  BNE.S      L000040
00000036 :  CMPI.L     #$75000002,10(A4)
0000003E :  BEQ.S      L000042
00000040 :  TRAP       #5
00000042 :  MOVEA.L    (A5),A3
00000044 :  MOVE.L     A3,D7
00000046 :  BEQ.S      L000054
00000048 :  CMPI.L     #$75000001,2(A3)
00000050 :  BEQ.S      L000054
00000052 :  TRAP       #5
00000054 :  MOVE.W     8(A3),D7
00000058 :  EXT.L      D7
0000005A :  MOVE.L     D7,14(A4)
```

bei 0000002C wird zunächst geprüft, ob p^ die Erweiterung B ist. Erst

dann kann geprüft werden, ob p^ auch noch C ist. Für q(B) ist dagegen nur eine Überprüfung nötig.

Automatisch abgefangene Fehler:

Für Fehler wie eine Division durch 0 oder das Verwenden von Zeigern auf ungerade Adressen bei Wort- oder Langwortzugriff wird kein Überprüfungscode erzeugt. Diese Fehler erzeugen automatisch einen Ausnahmezustand, der von NoGuru abgefangen wird.

13. Die Modulbibliothek:

Die Definitionen der Module wurden alle mit dem Programm ModToDef aus den Original-Quelltexten erzeugt.

Arguments:

```
DEFINITION Arguments;

IMPORT
  d : Dos;

VAR
  Me : d.ProcessPtr;

PROCEDURE NumArgs(): INTEGER;
PROCEDURE GetArg(arg: INTEGER; VAR argument: ARRAY OF CHAR);
PROCEDURE GetLock(num: INTEGER): d.FileLockPtr;

END Arguments.
```

Dieses Modul bietet Prozeduren zum Auslesen von CLI- und Workbenchargumenten.

CLI-Argumente sind diejenigen Namen, die beim Starten eines Programms von der CLI-Oberfläche aus hinter dem Programmnamen angegeben werden. Die verschiedenen Argumente werden durch Leerzeichen getrennt. Argumente, die selbst Leerzeichen enthalten, müssen dabei in Anführungszeichen geschrieben werden.

Workbenchargumente sind Icons, die zusätzlich zu dem gestarteten Programm, zusammen mit der Shift-Taste, angeklickt wurden. Workbenchargumente können einen leeren Namen haben, wenn Directory- oder Diskettenicons angeklickt wurden.

Das Modul Arguments behandelt CLI- und Workbenchargumente sehr ähnlich. Dadurch können Programme, die dieses Modul benutzen, meist vom CLI und von der Workbench aus benutzt werden.

Die Prozeduren im einzelnen:

PROCEDURE NumArgs(): INTEGER;

Gibt die Anzahl der Argumente zurück. Wurden keine Argumente angegeben, ist sie 0.

PROCEDURE GetArg(arg: INTEGER;
VAR argument: ARRAY OF CHAR);

Diese Prozedur dient zum Auslesen von Argumenten. Der erste Parameter gibt die Nummer des Argumentes an. Ist sie Null, wird der Name des gestarteten Programms zurückgegeben. Der zweite Parameter ist eine Zeichenkette, in die das Argument kopiert wird.

Bei Workbenchargumenten ändert GetArg() zusätzlich das aktive Verzeichnis (current directory) auf das Directory, in dem sich das Argument befindet.

PROCEDURE GetLock(num: INTEGER): d.FileLockPtr;

Diese Funktion gibt den Lock auf das Verzeichnis eines Argumentes zurück. Dieser Lock wird nicht dupliziert, darf also nicht mit Dos.UnLock() freigegeben werden. Bei CLI-Argumenten ist das Ergebnis immer der Lock des aktiven Verzeichnisses, nur bei Workbenchargumenten können es verschiedene Verzeichnisse sein.

ASCII:

```
DEFINITION ASCII;

CONST
  nul = 00X; soh = 01X; stx = 02X; etx = 03X;
  eot = 04X; enq = 05X; ack = 06X; bel = 07X;
  bs  = 08X; ht  = 09X; lf  = 0AX; vt  = 0BX;
  ff  = 0CX; cr  = 0DX; so  = 0EX; si  = 0FX;
  dle = 10X; dc1 = 11X; dc2 = 12X; dc3 = 13X;
  dc4 = 14X; nak = 15X; syn = 16X; etb = 17X;
  can = 18X; em  = 19X; sub = 1AX; esc = 1BX;
  fs  = 1CX; gs  = 1DX; rs  = 1EX; us  = 1FX;
  sp  = 20X; del = 7FX; eol =  lf; eof =  fs;
  csi = 9BX;

END ASCII.
```

Dieses implementationslose Modul enthält lediglich Konstanten für die Steuerzeichen des ASCII-Zeichencodes (American Standard Code for Information Interchange).

csi ist kein ASCII-Zeichen. Es wird vom Amiga benutzt um Kontrollsequenzen einzuleiten. Genaueres zu deren Aufbau ist [cbm:lb] zu entnehmen.

AVL:

```
DEFINITION AVL;

TYPE
  NodePtr = POINTER TO Node;
  Node = RECORD
    l : NodePtr;
    r : NodePtr;
    bal : INTEGER;
  END;
  Root = RECORD
    root : NodePtr;
    cmp : PROCEDURE(a, b: NodePtr): INTEGER;
    find : PROCEDURE(VAR root: Root; a: NodePtr): INTEGER;
  END;
  CompProc = PROCEDURE(a, b: NodePtr): INTEGER;
  FindProc = PROCEDURE(VAR root: Root; a: NodePtr): INTEGER;
  DoProc = PROCEDURE(a: NodePtr);

CONST
  left = -1;
  ok = 0;
  right = 1;

TYPE
  String = ARRAY 80 OF CHAR;
  SNodePtr = POINTER TO SNode;
  SNode = RECORD (Node)
    name : String;
  END;
  SRoot = RECORD (Root) END;

PROCEDURE Init(VAR root: Root; cmp: CompProc; find: FindProc);
PROCEDURE Add(VAR root: Root; node: NodePtr): BOOLEAN;
PROCEDURE Find(VAR root: Root): NodePtr;
PROCEDURE Remove(VAR root: Root; VAR node: NodePtr): BOOLEAN;
PROCEDURE DoForward(root: Root; proc: DoProc);
PROCEDURE DoBackward(root: Root; proc: DoProc);
PROCEDURE Dispose(VAR root: Root);
PROCEDURE SInit(VAR root: SRoot);
PROCEDURE SAdd(VAR root: SRoot; node: SNodePtr): BOOLEAN;
PROCEDURE SFind(VAR root: SRoot; str: String): SNodePtr;

END AVL.
```

Dieses Modul enthält Typen, die Knoten und Wurzel von AVL-Bäumen darstellen, und Prozeduren, die es erleichtern, mit diesen Bäumen zu arbeiten.

AVL-Bäume sind ausgeglichene binäre Bäume. Durch eine geschickte Art, den Baum auszugleichen, wird der Aufwand beim Einfügen und Entfernen von Elementen gering gehalten. Durch die Ausgeglichenheit werden Elemente in dem Baum sehr schnell gefunden. AVL-Bäume sind also ein guter Kompromiß zwischen unausgeglichenen und vollständig ausgeglichenen binären Bäumen. Eine genauere Beschreibung der AVL-Bäume kann [nw:aud] entnommen werden.

Um den Inhalt der Knoten und der Wurzel braucht man sich normalerweise nicht zu kümmern. Es reicht, wenn man die Prozeduren kennt und diese zur Manipulation des Baumes benutzt.

PROCEDURE Init(VAR root: Root;
cmp: CompProc;
find: FindProc);

Diese Prozedur initialisiert und leert den AVL-Baum mit der Wurzel root. cmp ist eine Prozedur, die 2 Knoten vergleicht. Dabei soll sie beim Aufruf cmp(a,b) folgende Ergebnisse liefern:

wenn a < b: Ergebnis < 0
wenn a = b: Ergebnis = 0
wenn a > b: Ergebnis > 0

Die Prozedur *find* wird dazu benötigt, ein Element im Baum mit der Prozedur Find() (s.u.) zu suchen. Sie funktioniert ähnlich wie *cmp*. Beim Aufruf find(root,a) sollen folgende Ergebnisse geliefert werden:

wenn a < dem gesuchten Element: Ergebnis < 0
wenn a = dem gesuchten Element: Ergebnis = 0
wenn a > dem gesuchten Element: Ergebnis > 0.

PROCEDURE Add(VAR root: Root;
node: NodePtr): BOOLEAN;

Add fügt das neue Element node in den Baum ein. Das Ergebnis ist TRUE, wenn das Einfügen erfolgreich war und FALSE, wenn *node* nicht mehr eingefügt werden konnte. Dies ist nur dann der Fall, wenn ein gleiches Element (siehe Prozedur *cmp* bei Init()), bereits im Baum enthalten war.

PROCEDURE Find(VAR root: Root): NodePtr;

Mit Find können Elemente im Baum gesucht werden. Dabei ist das Ergebnis ein Zeiger auf das Element, bei dem die Prozedur find(), die Init() übergeben wurde, 0 als Ergebnis liefert. Wurde kein passendes Element gefunden, ist das Ergebnis NIL.

PROCEDURE Remove(VAR root: Root;
VAR node: NodePtr): BOOLEAN;

Remove entfernt das Element *node* aus dem Baum. Sein Speicher muß danach noch mit DISPOSE() freigegeben werden. Das Ergebnis ist TRUE, wenn *node* entfernt wurde, und FALSE, wenn *node* nicht im Baum enthalten war.

PROCEDURE DoForward(root: Root; proc: DoProc);

DoForward ruft *proc* mit jedem Element des Baumes als Parameter auf. Dabei werden die Elemente von 'links nach rechts' aufgerufen, d.h. das kleinste Element zuerst und das größte zuletzt.

PROCEDURE DoBackward(root: Root; proc: DoProc);

Diese Prozedur funktioniert ähnlich wie *DoForward*, durchläuft den Baum jedoch 'rückwärts', so daß *proc* zunächst mit dem größten und zum Schluß mit dem kleinsten Element aufgerufen wird.

PROCEDURE Dispose(VAR root: Root);

Dispose entfernt alle Elemente aus dem Baum und gibt ihren Speicher frei.

String-AVL-Bäume:

In vielen Fällen braucht man einen AVL-Baum dazu, Zeichenketten alphabetisch sortiert zu speichern. Um dies zu erleichtern, bietet dieses Modul die Typen SNode und SRoot, die Knoten und Wurzeln von String-Bäumen darstellen. SNode.name enthält dabei immer den Namen eines Knotens. Folgende Prozeduren zum Arbeiten mit String-Bäumen gibt es:

PROCEDURE SInit(VAR root: SRoot);

Dies ist das Gegenstück zu Init(). Es initialisiert einen String-AVL-Baum mit der Wurzel root. Die Prozeduren *cmp* und *find*, die bei Init() übergeben werden müssen, werden auf 2 interne Prozeduren des Moduls AVL gesetzt. Man braucht sich also nicht weiter darum zu kümmern.

PROCEDURE SAdd(VAR root: SRoot;
node: SNodePtr): BOOLEAN;

Fügt *node* in einen String-AVL-Baum ein. SAdd entspricht der Prozedur Add() (s.o.)

PROCEDURE SFind(VAR root: SRoot;
str: String): SNodePtr;

SFind durchsucht den String-AVL-Baum nach einem Knoten mit dem Namen str und gibt seine Adresse oder bei Mißérfolg NIL zurück.

Die Prozeduren Remove, DoForward, DoBackward und Dispose können ohne Änderungen auch für String-Bäume verwendet werden.

Beispielprogramm:

```
MODULE Bank;

IMPORT AVL, io;

TYPE BankNode = RECORD (AVL.Node)
                  nummer: INTEGER;
                  guthaben: LONGINT;
                END;
VAR
  root: AVL.Root;
  bnode: POINTER TO BankNode;
  node: AVL.NodePtr;
  str: ARRAY 80 OF CHAR;
  ch: CHAR;
  nummer: INTEGER;
  betrag: LONGINT;

  PROCEDURE * Cmp(a,b: AVL.NodePtr): INTEGER;
  BEGIN RETURN a(BankNode).nummer - b(BankNode).nummer END Cmp;

  PROCEDURE * Find(VAR root: AVL.Root; node: AVL.NodePtr): INTEGER;
  BEGIN RETURN node(BankNode).nummer - nummer END Find;

BEGIN
  AVL.Init(root,Cmp,Find);
  LOOP
    io.WriteString("Ein-, Auszahlung, Konto eröffnen, schließen, Ende");
    io.WriteString(" (E/A/O/S/Q)? "); io.ReadString(str); ch := CAP(str[0]);
    CASE ch OF
    "Q": EXIT |
    "A","E","O","S":
      REPEAT io.WriteString("Kontonummer: ")
      UNTIL io.ReadInt(betrag) AND (betrag>=0) AND (betrag<=MAX(INTEGER));
      nummer := SHORT(betrag);
      IF ch="O" THEN
        NEW(bnode); IF bnode=NIL THEN HALT(20) END;
        bnode.nummer := nummer;
        REPEAT io.WriteString("Guthaben : ") UNTIL io.ReadInt(bnode.guthaben);
        IF NOT AVL.Add(root,bnode) THEN
          io.WriteString("Kontonummer existiert schon!"); io.WriteLn;
        END;
      ELSE
        node := AVL.Find(root);
        IF node=NIL THEN
          io.WriteString("Konto nicht gefunden!"); io.WriteLn;
```

```
      ELSE
        IF ch="S" THEN
          IF AVL.Remove(root,node) THEN END;
          DISPOSE(node);
        ELSE
          WITH node: BankNode DO
            REPEAT io.WriteString("Betrag: ") UNTIL io.ReadInt(betrag);
            IF ch="A" THEN DEC(node.guthaben,betrag)
                      ELSE INC(node.guthaben,betrag) END;
            io.WriteString("neuer KontoStand: ");
            io.WriteInt(node.guthaben,8); io.WriteLn;
          END;
        END;
      END;
    END;
  ELSE END;  (* CASE ch OF *)
 END;  (* LOOP *)
END Bank.
```

Dieses Programm verwaltet einen AVL-Baum, der Bankkonten nach ihren Kontonummern sortiert verwaltet. Es ermöglicht das Eröffnen neuer Konten, das Schließen von Konten und das Ein- und Auszahlen von Beträgen. Erweiterungen des Programms, z.B. um eine Funktion zum Aufschlagen der Zinsen am Jahresende auf alle Konten, sind leicht mit *DoForward* möglich.

Beep:

```
DEFINITION Beep;

PROCEDURE Beep(low: BOOLEAN);

END Beep.
```

Mit diesem kurzen Modul können einfache Pieptöne erzeugt werden. Es ist oft sinnvoll, daß Programme den Benutzer mit akustischen Signalen 'aufwecken', um ihn auf Fehler aufmerksam zu machen oder ihm anzuzeigen, daß ein längerer Prozeß gerade beendet wurde und das Programm nun auf neue Eingaben wartet.

Die Prozedur *Beep* dieses Moduls erzeugt einen solchen Ton. Je nachdem, ob der Parameter TRUE oder FALSE ist, wird ein tiefer, warnender Ton oder ein höherer Signalton erzeugt.

Break:

```
DEFINITION Break;

PROCEDURE CtrlCOff;
PROCEDURE CtrlCOn;

END Break.
```

Programme, die dieses Modul importieren, können mit der Tastenkombination Control und C abgebrochen werden. Beim Abbruch wird die Meldung "*** Break" ausgegeben.

Alle guten Programme sollten dieses Modul importieren, um dem Benutzer eine einfache Möglichkeit zum Abbruch des Programms zu bieten. Außerdem ist Break während der Testphase nützlich, um in Endlosschleifen hängengebliebene Programme zu stoppen.

Mit den Prozeduren *CtrlCOff()* und *CtrlCOn()* kann die Abbruchmöglichkeit kurzzeitig aus und wieder eingeschaltet werden. Es ist in manchen Situationen nötig, sicherzustellen, daß ein Programm im Augenblick nicht abgebrochen wird.

BreakRq:

```
DEFINITION BreakRq;

PROCEDURE CtrlCOff;
PROCEDURE CtrlCOn;

END BreakRq.
```

BreakRq leistet das gleiche wie Break. Es benutzt jedoch nicht das Modul io, um den Text "*** Break" auszugeben, sondern öffnet mit Hilfe von *Assert()* aus *Requests* einen Requester. Der Vorteil ist, daß so *io* nicht importiert wird und beim Start von der Workbench das io-Fenster nicht geöffnet wird. Nachteilig ist, daß beim Arbeiten im CLI der Requester mit der Maus angeklickt werden muß und man so die Hände von der Tastatur nehmen muß.

Conversions:

```
DEFINITION Conversions;

PROCEDURE StringToInt(VAR str: ARRAY OF CHAR;
                      VAR int: LONGINT): BOOLEAN;
PROCEDURE IntToString(   int: LONGINT;
                      VAR str: ARRAY OF CHAR;
                      n: INTEGER): BOOLEAN;
PROCEDURE IntToHex   (   int: LONGINT;
                      VAR str: ARRAY OF CHAR;
                      n: INTEGER): BOOLEAN;
PROCEDURE StrToInt(VAR str: ARRAY OF CHAR;
                      VAR int: LONGINT;
                      base: INTEGER): BOOLEAN;
PROCEDURE IntToStr(int: LONGINT;
                      VAR str: ARRAY OF CHAR;
                      base, width: SHORTINT;
                      fillChar: CHAR): BOOLEAN;

END Conversions.
```

Mit den Prozeduren dieses Moduls können Zeichenketten in INTEGER-Zahlen umgewandelt werden und umgekehrt.

PROCEDURE StringToInt(VAR str: ARRAY OF CHAR; VAR int: LONGINT): BOOLEAN;

wandelt die Zeichenkette *str* in eine LONGINT-Zahl um. Dabei kann *str* eine vorzeichenbehaftete LONGINT-Zahl oder eine Hex-Zahl, die von einem "H" gefolgt wird, sein. Das Ergebnis ist TRUE, wenn *str* eine korrekte Zahl enthielt oder leer war.

PROCEDURE IntToString(int: LONGINT; VAR str: ARRAY OF CHAR; n: INTEGER): BOOLEAN;

Wandelt *int* in eine *n*-stellige Dezimalzahl um. Die Zeichenkette *str* muß mindesten 2+n Zeichen lang sein. Das Ergebnis ist FALSE, wenn *int* mehr als *n* Stellen benötigt.

PROCEDURE IntToHex(int: LONGINT;
VAR str: ARRAY OF CHAR;
n: INTEGER): BOOLEAN;

Wandelt *int* analog zu *IntToString* in eine Hexadezimalzahl mit *n* Stellen um.

PROCEDURE StrToInt(VAR str: ARRAY OF CHAR;
VAR int: LONGINT;
base: INTEGER): BOOLEAN;

Wandelt den String *str* in eine Integerzahl um. Dabei gibt *base* das Zahlensystem an, in dem *str* übergeben wird (z.B. 2 für Binärsystem, 16 für Hexadezimalsystem). Das Ergebnis ist TRUE, wenn eine korrekte Zahl übergeben wurde.

PROCEDURE IntToStr(int: LONGINT;
VAR str: ARRAY OF CHAR;
base, width: SHORTINT;
fillChar: CHAR): BOOLEAN;

Wandelt die Integerzahl *int* in einen String um. Dabei gibt *base* das Zahlensystem und *width* die Breite des Strings an. Die Zahl wird rechtsbündig in *str* geschrieben. Die unbenutzten Stellen vor dem String werden mit *fillChar* gefüllt. Das Ergebnis ist bei Erfolg TRUE, d.h. wenn die Zahl in die zur Verfügung stehenden Stellen gepaßt hat.

Display:

```
DEFINITION Display;

IMPORT I: Intuition, g: Graphics, li: Lists;

TYPE
  DispEl = RECORD (li.Node)
    title : POINTER TO ARRAY 80 OF CHAR;
    rp : g.RastPortPtr;
    font : g.TextFontPtr;
    width : INTEGER;
    height : INTEGER;
    turtleX : REAL;
    turtleY : REAL;
    turtleDir : REAL;
    pen : BOOLEAN;
    cursor : BOOLEAN;
    curX : INTEGER;
    curY : INTEGER;
    curXAbs : INTEGER;
    curYAbs : INTEGER;
    txtWidth : INTEGER;
    txtHeight : INTEGER;
  END;
  Screen = RECORD (DispEl)
    screen : I.ScreenPtr;
    li : g.LayerInfoPtr;
    layer : g.LayerPtr;
  END;
  Window = RECORD (DispEl)
    window : I.WindowPtr;
  END;
  DispElPtr = POINTER TO DispEl;
  ScreenPtr = POINTER TO Screen;
  WindowPtr = POINTER TO Window;

CONST   (* Konstanten für LinePattern *)
  line = -1;
  dots = 5555H;
  bigdots = 3333H;
  broken = 0F0FH;

PROCEDURE OpenScreen (scrn: ScreenPtr;
                      title: ARRAY OF CHAR;
                      x, y, w, h: INTEGER;
                      d: SHORTINT;
```

```
                    hires, lace: BOOLEAN): BOOLEAN;
PROCEDURE OpenWindow (win: WindowPtr;
                    title: ARRAY OF CHAR;
                    x, y, w, h: INTEGER;
                    screen: I.ScreenPtr): BOOLEAN;
PROCEDURE Close (d: DispElPtr);
PROCEDURE Init (d: DispElPtr);
PROCEDURE SetColors( s: ScreenPtr;  VAR cols: ARRAY OF INTEGER);
PROCEDURE SetCol (s: ScreenPtr; num, R, G, B: INTEGER);
PROCEDURE NumToRGB (num: INTEGER; VAR r, g, b: INTEGER);
PROCEDURE RGBToNum (r, g, b: INTEGER): INTEGER;
PROCEDURE FrontPen (d: DispElPtr; pen: SHORTINT);
PROCEDURE BackPen (d: DispElPtr; pen: SHORTINT);
PROCEDURE Jam1 (d: DispElPtr);
PROCEDURE Jam2 (d: DispElPtr);
PROCEDURE Complement (d: DispElPtr);
PROCEDURE LinePattern (d: DispElPtr; pat: INTEGER);
PROCEDURE Clear (d: DispElPtr);
PROCEDURE Line (d: DispElPtr; x1, y1, x2, y2: INTEGER);
PROCEDURE Dot (d: DispElPtr; x, y: INTEGER);
PROCEDURE Rect (d: DispElPtr; x, y, w, h: INTEGER);
PROCEDURE Box (d: DispElPtr; x, y, w, h: INTEGER);
PROCEDURE Move (d: DispElPtr; x, y: INTEGER);
PROCEDURE Draw (d: DispElPtr; x, y: INTEGER);
PROCEDURE Text (d: DispElPtr; x, y: INTEGER; s: ARRAY OF CHAR);
PROCEDURE Circle (d: DispElPtr; x, y, r: INTEGER);
PROCEDURE Ellipse (d: DispElPtr; x, y, rx, ry: INTEGER);
PROCEDURE Font (d: DispElPtr;
                    name: ARRAY OF CHAR;
                    height: INTEGER): BOOLEAN;
PROCEDURE SetTurtlePos (d: DispElPtr; x, y: REAL);
PROCEDURE GetTurtlePos (d: DispElPtr; VAR x, y: REAL);
PROCEDURE SetTurtleDir (d: DispElPtr; dir: REAL);
PROCEDURE GetTurtleDir (d: DispElPtr): REAL;
PROCEDURE SetPen (d: DispElPtr);
PROCEDURE LiftPen (d: DispElPtr);
PROCEDURE Forward (d: DispElPtr; s: REAL);
PROCEDURE TurnLeft (d: DispElPtr; alpha: REAL);
PROCEDURE TurnRight (d: DispElPtr; alpha: REAL);
PROCEDURE SetCursor (d: DispElPtr; on: BOOLEAN): BOOLEAN;
PROCEDURE CursorOn (d: DispElPtr);
PROCEDURE CursorOff (d: DispElPtr);
PROCEDURE Position (d: DispElPtr; x, y: INTEGER);
PROCEDURE Plain (d: DispElPtr);
PROCEDURE UnderLinedOn (d: DispElPtr);
PROCEDURE UnderLinedOff (d: DispElPtr);
PROCEDURE BoldOn (d: DispElPtr);
PROCEDURE BoldOff (d: DispElPtr);
```

```
PROCEDURE ItalicOn (d: DispElPtr);
PROCEDURE ItalicOff (d: DispElPtr);
PROCEDURE Home (d: DispElPtr);
PROCEDURE ClrHome (d: DispElPtr);
PROCEDURE ScrollUp (d: DispElPtr);
PROCEDURE ScrollDown (d: DispElPtr);
PROCEDURE InsertLine (d: DispElPtr; n: INTEGER);
PROCEDURE DeleteLine (d: DispElPtr; n: INTEGER);
PROCEDURE WriteLn (d: DispElPtr);
PROCEDURE Write (d: DispElPtr; Str: ARRAY OF CHAR);

END Display.
```

Dieses mächtige Modul beinhaltet eine große Anzahl von Zeichenprozeduren, auch Turtlegraphics und Prozeduren zur komfortablen Textausgabe. Das Modul wurde z.B. benutzt, um den Editor OEd zu schreiben.

Die Prozeduren können fast alle auf Screens und auf Windows angewendet werden. Dadurch sind sie sehr flexibel einsetzbar.

Prozeduren zum Öffnen und Schließen von Screens und Windows:

```
PROCEDURE OpenScreen(scrn: ScreenPtr;
                     title: ARRAY OF CHAR;
                     x, y, w, h: INTEGER;
                     d: SHORTINT;
                     hires, lace: BOOLEAN): BOOLEAN;
```

Öffnet einen Screen. *scrn* ist ein Zeiger auf ein Screen-Record. Der Speicher dafür muß vorher mit NEW() alloziiert werden. *title* ist der Titel des Screens. Ist er leer ("""), wird ein Screen ohne Titelzeile erzeugt. *x* und *y* geben die Position (normalerweise 0/0) des Screens, *w* und *h* seine Breite bzw. Höhe an.

d ist die Tiefe des Screens, d.h. die Anzahl der Bitplanes, die für den

Screen alloziert werden. d darf normalerweise höchstens 5, bei hires maximal 4 sein. Die Anzahl der Farben, die auf dem Screen zur Verfügung stehen, ist 2 hoch d.

hires und *lace* geben an, ob ein Hires- bzw. Interlace-Screen geöffnet werden soll. Bei Hires verdoppelt sich die horizontale, bei Interlace die vertikale Auflösung.

Das Ergebnis von *OpenScreen* ist TRUE, wenn der Screen geöffnet werden konnte, sonst FALSE.

```
PROCEDURE OpenWindow(win: WindowPtr;
                     title: ARRAY OF CHAR;
                     x, y, w, h: INTEGER;
                     screen: I.ScreenPtr): BOOLEAN;
```

Mit *OpenWindow()* kann analog zu *OpenScreen()* ein Fenster geöffnet werden. *win* zeigt auf das Window-Record und muß zuvor alloziiert werden. *title* gibt den Titel des Fensters an. Der Titel kann nicht wie bei *OpenScreen()* durch einen leeren String unterdrückt werden. *x, y* sind die Position, *w* und *h* die Breite und Höhe des Fensters.

Soll das Fenster auf der Workbench geöffnet werden, kann *screen* einfach auf NIL gesetzt werden. Soll es dagegen auf einem anderen Screen geöffnet werden, muß ein Zeiger auf die Intuition-Screenstruktur übergeben werden. Diese steht in *Screen.screen.*

Das Fenster wird mit Größen-, Verschiebungs-, Tiefen- und Schließgadgets ausgestattet. Um das Größen und das Schließgadget abzufragen, müssen die IDCMPFlags noch mit

```
Intuition.ModifyIDCMP(win.window,
  LONGSET{Intuition.newSize,Intuition.closeWindow});
```

gesetzt werden. Dann können mit Exec.GetMsg(win.window.userPort) Messages über erfolgte Ereignisse geholt werden. Genaueres

über die IDCMPMessages kann [cbm:airm] entnommen werden.

Das Ergebnis von *OpenWindow()* ist bei Erfolg TRUE, sonst FALSE.

PROCEDURE Close(d: DispElPtr);

Close kann zum Schließen von Screens und Windows verwendet werden. Dabei wird aller mit dem Screen oder Window verbundene Speicher für Fonts etc. freigegeben.

PROCEDURE Init(d: DispElPtr);

Init muß bei Fenstern aufgerufen werden, wenn sich deren Größe geändert hat, um die Breite und Höhe des Fensters neu zu setzen und die neue Textbreite und -höhe auszurechnen.

Farben im Screen setzen:

PROCEDURE SetColors(s: ScreenPtr;
VAR cols: ARRAY OF INTEGER);

Mit SetColors können die Farben eines Screens neu gesetzt werden. cols ist ein Feld, das für jede Farbe einen INTEGER-Wert enthält. Dieser Wert ist die Summe aus 256 mal dem Rotanteil, 16 mal dem Grünanteil und dem Blauanteil der Farbe. In hexadezimaler Schreibweise geben die letzten 3 Ziffern die Rot-, Grün- und Blauwerte an. Beispiel: pink = 0F0FH.

PROCEDURE SetCol(s: ScreenPtr;
num, R, G, B: INTEGER);

SetCol setzt die Farbe num auf die Rot-, Grün- und Blauwerte R, G und B. Die Farben sind mit 0 beginnend durchnummeriert.

PROCEDURE NumToRGB(num: INTEGER;
VAR r, g, b: INTEGER);
PROCEDURE RGBToNum(r, g, b: INTEGER): INTEGER;

Mit diesen beiden Prozeduren können Farbnummern, wie sie bei *SetColors* übergeben werden, in ihre Rot-, Grün- und Blauanteile zerlegt werden, und umgekehrt können die Farbwerte in Farbnummern umgewandelt werden.

Zeichenmodi und Zeichenfarben:

PROCEDURE FrontPen(d: DispElPtr; pen: SHORTINT);

Setzt die Zeichenfarbe auf Farbe Nummer *pen*.

PROCEDURE BackPen(d: DispElPtr; pen: SHORTINT);

Setzt die Hintergrundfarbe. Sie wird nur bei Textausgabe mit dem Zeichenmodus Jam2 verwendet.

PROCEDURE Jam1(d: DispElPtr);

Aktiviert den Zeichenmodus Jam1. Dies ist der normale Modus zum Zeichnen mit der Farbe, die mit *FrontPen()* gesetzt wurde.

PROCEDURE Jam2(d: DispElPtr);

Aktiviert den Zeichenmodus Jam2. Jam2 verwendet sowohl den Front- als auch den BackPen. Dies ist bei der Textausgabe nötig, wenn man z.B. anderen Text überschreiben will.

PROCEDURE Complement(d: DispElPtr);

Setzt den Zeichenmodus Complement. Er bewirkt, daß immer mit der logischen Komplementärfarbe des Hintergrundes gezeichnet wird. Die logische Komplementärfarbe zur Farbe Nummer x ist bei insgesamt n

Farben die Farbe mit der Nummer n-1-x. So ist z.B. bei 8 Farben die Komplementärfarbe zu Farbe 2 die Farbe 5.

Der große Vorteil des Komplementärmodus ist, daß man das Gezeichnete sehr leicht wieder rückgängig machen kann, indem man es einfach noch einmal zeichnet. Die Komplementärfarbe der Komplementärfarbe ist nämlich wieder die ursprüngliche Farbe. Daher werden Dinge wie Fadenkreuze oder Ausschneiderechtecke meist mit diesem Modus gezeichnet.

PROCEDURE LinePattern(d: DispElPtr; pat: INTEGER);

Beim Zeichnen von Linien kann man ein Muster benutzen. Normalerweise hat es den Wert 'line' (siehe Konstanten in der Definition). Dieses Muster erzeugt durchgezogene Linien. Mit LinePattern() kann man noch andere Muster setzen. Ein paar oft benötigte Werte sind als Konstanten definiert:

line	: durchgezogene Linie
dots	: gepunktete Linie
bigdots	: mit kleinen Strichen gestrichelte Linie
broken	: gestrichelte Linie

Einfache Zeichenroutinen:

PROCEDURE Clear(d: DispElPtr);

Löscht den Inhalt des Fensters oder Screens.

PROCEDURE Line(d: DispElPtr; x1, y1, x2, y2: INTEGER);

Zeichnet eine Linie von (x1;y1) nach (x2;y2).

PROCEDURE Dot(d: DispElPtr; x, y: INTEGER);

Setzt einen Punkt an der Position (x;y).

PROCEDURE Rect(d: DispElPtr; x, y, w, h: INTEGER);

Zeichnet ein Rechteck mit der linken oberen Ecke an (*x*;*y*), der Breite *w* und der Höhe *h*.

PROCEDURE Box(d: DispElPtr; x, y, w, h: INTEGER);

Zeichnet ein ausgefülltes Rechteck mit der linken oberen Ecke an Position (*x*;*y*), der Breite *w* und der Höhe *h*.

PROCEDURE Move(d: DispElPtr; x, y: INTEGER);
PROCEDURE Draw(d: DispElPtr; x, y: INTEGER);

Move bewegt einen (unsichtbaren) Cursor an die Position (*x*;*y*). Draw zeichnet dann von dieser Position aus eine Line nach (*x*;*y*) und setzt den Cursor auf die neue Position. Mit *Move()* und *Draw()* können leicht durchgehende Linienzüge gezeichnet werden.

PROCEDURE Text(d: DispElPtr;
x, y: INTEGER;
s: ARRAY OF CHAR);

Schreibt die Zeichenkette s an die Position (x;y).

PROCEDURE Circle(d: DispElPtr; x, y, r: INTEGER);

Circle() zeichnet einen Kreis mit dem Mittelpunkt (*x*;*y*) und dem Radius *r*.

PROCEDURE Ellipse(d: DispElPtr; x, y, rx, ry: INTEGER);

Zeichnet eine Ellipse mit dem horizontalen Radius *rx* und dem vertikalen Radius *ry*. Die restlichen Parameter entsprehen denen von *Circle()*.

PROCEDURE Font(d: DispElPtr;
name: ARRAY OF CHAR;
height: INTEGER): BOOLEAN;

Setzt den Zeichensatz für die Textausgabe. *name* ist sein Name und *height* seine Höhe. Das Ergebnis ist TRUE, wenn der Font gefunden wurde und geladen werden konnte.

Beispiel: IF NOT Font(w,"topaz.font",9) THEN Error END;

Turtlegraphic Routinen:

Turtlegraphic kann man sich sehr anschaulich so vorstellen, daß eine Schildkröte mit einem Stift im Maul über den Bildschirm wandert. Diese Schildkröte kann nur vorwärts laufen oder sich nach rechts oder links drehen. Durch diese elementaren Bewegungen kann man manche Grafiken, wie Hilbert-Kurven etc., sehr einfach erzeugen.

PROCEDURE SetTurtlePos(d: DispElPtr; x, y: REAL);
PROCEDURE GetTurtlePos(d: DispElPtr; VAR x, y: REAL);

Setzt die Schildkröte an die Position (x;y) bzw. liest ihre aktuelle Position aus.

PROCEDURE SetTurtleDir(d: DispElPtr; dir: REAL);
PROCEDURE GetTurtleDir(d: DispElPtr): REAL;

Setzt bzw. holt die Richtung, in die die Schildkröte läuft. Dabei bedeutet 0° nach oben, 90° nach links, 180° nach unten etc.

PROCEDURE SetPen(d: DispElPtr);
PROCEDURE LiftPen(d: DispElPtr);

Mit diesen Prozeduren wird die Schildkröte veranlaßt, ihren Stift abzusetzen, also das Zeichnen zu beginnen, oder ihn wieder hochzunehmen, also nicht mehr zu zeichnen.

PROCEDURE Forward(d: DispElPtr; s: REAL);

Dies ist die einzige Prozedur bei Turtlegraphics, die wirklich etwas zeichnet. Die Schildkröte wandert eine Strecke der Länge *s* (in Pixel) geradeaus und zeichnet dabei eine Linie, wenn ihr Stift gerade gesetzt war.

PROCEDURE TurnLeft(d: DispElPtr; alpha: REAL);
PROCEDURE TurnRight(d: DispElPtr; alpha: REAL);

Diese beiden Prozeduren drehen die Schildkröte um *alpha* Grad nach links bzw. rechts.

Komfortable Textausgabe:

Mit den folgenden Prozeduren kann Text wie bei einer Console ausgegeben werden. Dabei wird automatisch in die jeweils nächste Zeile gesprungen, wenn man Text über den rechten Fensterrand ausgibt und der Text wird hochgescrollt, wenn in der untersten Zeile ein WriteLn ausgeführt wird. Zudem kann optional ein rechteckiger Cursor angezeigt werden.

PROCEDURE SetCursor(d: DispElPtr;
on: BOOLEAN): BOOLEAN;

Schaltet den Cursor je nachdem ob *on* TRUE oder FALSE ist, ein oder aus. Das Ergebnis ist der vorige Zustand des Cursor. Damit kann der Cursor kurzzeitig ausgeschaltet, und danach sein vorheriger Zustand wieder hergestellt werden, wenn das Ergebnis dieser Prozedur in einer Variablen zwischengespeichert wird.

PROCEDURE CursorOn(d: DispElPtr);
PROCEDURE CursorOff(d: DispElPtr);

Diese Prozeduren schalten den Cursor ein bzw. aus. Dabei ist es egal, ob der Cursor vorher ein- oder ausgeschaltet war.

PROCEDURE Position(d: DispElPtr; x, y: INTEGER);

Setzt den Cursor an die Position (*x*;*y*). Die Position wird nicht in Pixel, sondern in Zeichen angegeben. Die Anzahl der Zeichen je Zeile und Spalte hängt vom verwendeten Zeichensatz ab. Wurde kein Zeichensatz mit *Font()* gesetzt, hängt dieser von den gerade aktiven Preferences ab.

Die Anzahl der Zeichen je Zeile und Spalte steht in DispEl.txtWidth bzw. DispEl.txtHeight. Der Cursor sollte nicht an Positionen gesetzt werden, die größer oder gleich diesen Werten sind.

PROCEDURE Plain(d: DispElPtr);

Schaltet alle besonderen Textattribute, wie unterstrichen, fett etc., ab.

PROCEDURE UnderLinedOn(d: DispElPtr);
PROCEDURE UnderLinedOff(d: DispElPtr);

Schalten unterstrichenen Text ein bzw. aus.

PROCEDURE BoldOn(d: DispElPtr);
PROCEDURE BoldOff(d: DispElPtr);

Schalten Fettschrift ein bzw. aus.

PROCEDURE ItalicOn(d: DispElPtr);
PROCEDURE ItalicOff(d: DispElPtr);

Schalten Kursivschrift ein bzw. aus.

PROCEDURE Home(d: DispElPtr);
PROCEDURE ClrHome(d: DispElPtr);

Setzen den Cursor nach (0,0). *ClrHome()* löscht zusätzlich den Inhalt des Fensters oder Screens.

PROCEDURE ScrollUp(d: DispElPtr);
PROCEDURE ScrollDown(d: DispElPtr);

Scrollt den Bildschirm eine Zeile hoch (up) oder herunter (down). Dabei bleibt die Cursorposition relativ zur linken oberen Ecke des Fensters oder Screens bestehen.

PROCEDURE InsertLine(d: DispElPtr; n: INTEGER);

Fügt an Zeile n eine Leerzeile ein und scrollt den Text darunter um eine Zeile nach unten.

PROCEDURE DeleteLine(d: DispElPtr; n: INTEGER);

Löscht Zeile n und scrollt den Text darunter um eine Zeile nach oben.

PROCEDURE WriteLn(d: DispElPtr);

Setzt den Cursor an die erste Position in die nächste Zeile.

PROCEDURE Write(d: DispElPtr; Str: ARRAY OF CHAR);

Gibt den Text *Str* an der aktuellen Cursorposition aus. Wird der rechte Rand erreicht, so wird der Text aufgeteilt und in der nächsten Zeile weitergeschrieben. Geschieht dies am unteren Bildschirmrand, wird der Text um eine Zeile hochgescrollt.

FileReq:

```
DEFINITION FileReq;

PROCEDURE FileReq(hail: ARRAY OF CHAR;
                  VAR name: ARRAY OF CHAR): BOOLEAN;

END FileReq.
```

Die Prozedur dieses Moduls öffnet den Filerequester der ARP-Library und bietet damit eine komfortable Möglichkeit, einen Dateinamen einzulesen.

PROCEDURE FileReq(hail: ARRAY OF CHAR; VAR name: ARRAY OF CHAR): BOOLEAN;

Öffnet den ARP-File-Requester. Damit dies funktioniert, muß sich die arp.library im LIBS: Directory des Computers befinden, auf dem das Programm gerade läuft. Dies ist leider nicht immer der Fall.

hail ist der Text für die Titelzeile des Requesters. Er sollte normalerweise den Programmnamen und die vorgenommene Aktion enthalten. *name* ist der Name der ausgewählten Datei inklusive des Pfades.

Das Ergebnis von FileReq ist TRUE, wenn der Benutzer "OK" gedrückt hat. Bei Cancel ist es FALSE.

Beispiel:

```
IF Requests.FileReq("Load:",name) THEN Load(name) END;
```

FileSystem:

```
DEFINITION FileSystem;

IMPORT
  d : Dos;

CONST
  ok = 0;
  eof = 1;
  readerr = 2;
  writeerr = 3;
  onlyread = 4;
  onlywrite = 5;
  toofar = 6;
  outofmem = 7;
  cantopen = 8;
  cantlock = 9;

TYPE
  FilePtr = POINTER TO File;
  File = RECORD
    handle : d.FileHandlePtr;
    status : INTEGER;
    write : BOOLEAN;
    name : ARRAY 256 OF CHAR;
  END;

PROCEDURE Open(VAR file: File;
                    name: ARRAY OF CHAR;
                    write: BOOLEAN): BOOLEAN;
PROCEDURE Close(VAR file: File): BOOLEAN;
PROCEDURE Read(VAR file: File;
                    VAR   to: ARRAY OF BYTE): BOOLEAN;
PROCEDURE ReadChar(VAR file: File;
                         VAR  ch: CHAR): BOOLEAN;
PROCEDURE ReadString(VAR file: File;
                          VAR   to: ARRAY OF CHAR): BOOLEAN;
PROCEDURE ReadBlock(VAR file: File;
                          to, size: LONGINT): BOOLEAN;
PROCEDURE Write(VAR file: File;
                    VAR from: ARRAY OF BYTE): BOOLEAN;
PROCEDURE WriteChar(VAR file: File;
                        ch: CHAR): BOOLEAN;
PROCEDURE WriteString(VAR file: File;
                          from: ARRAY OF CHAR): BOOLEAN;
PROCEDURE WriteBlock(VAR file: File;
```

```
                    from, size: LONGINT): BOOLEAN;
PROCEDURE Size(VAR file: File): LONGINT;
PROCEDURE Move(VAR file: File;
                    to: LONGINT): BOOLEAN;
PROCEDURE Forward(VAR file: File;
                    to: LONGINT): BOOLEAN;
PROCEDURE Backward(VAR file: File;
                    to: LONGINT): BOOLEAN;
PROCEDURE Delete(VAR file: File): BOOLEAN;
PROCEDURE Exists(name: ARRAY OF CHAR): BOOLEAN;

END FileSystem.
```

Dieses Modul erleichtert das Arbeiten mit Dateien. Es stellt Prozeduren zur Dateiverwaltung zur Verfügung. Die Geschwindigkeit dieses Moduls liegt normalerweise über der von Dos, da beim Lesen und Schreiben Puffer verwendet werden.

Um mit diesen Prozeduren zu arbeiten, muß zunächst eine Variable vom Typ File deklariert werden. Mit ihr kann dann mit *Open()* eine Datei geöffnet werden, die dann mit den anderen Prozeduren bearbeitet werden kann und zum Schluß mit *Close()* geschlossen wird.

Die Prozeduren, die ein boolesches Ergebnis liefern, waren immer erfolgreich, wenn sie TRUE zurückliefern. Ansonsten steht in dem Recordelement File.status eine Fehlernummer, die eine genauere Diagnose der Fehlerursache erlaubt.

Dateien, die nicht geschlossen wurden, werden automatisch beim Programmende geschlossen. Es ist jedoch schlechter Stil, sich darauf zu verlassen. Man sollte also alle Dateien selbst mit der Prozedur *Close()* schließen.

PROCEDURE Open(VAR file: File;
name: ARRAY OF CHAR;
write: BOOLEAN): BOOLEAN;

Öffnet die Datei mit dem Namen *name*. Ist *write* TRUE, so wird die Datei neu erzeugt und zum Schreiben geöffnet. Dann dürfen die Lese-

prozeduren nicht verwendet werden. Ist hingegen *write* FALSE, so darf in die Datei nicht geschrieben werden.

PROCEDURE Close(VAR file: File): BOOLEAN;

Schließt die mit Open() geöffnete Datei.

PROCEDURE Read(VAR file: File;
VAR to: ARRAY OF BYTE): BOOLEAN;

Liest LEN(to) Bytes aus der Datei und speichert sie in *to*. *to* können Variablen aller Typen zugewiesen werden. *Read()* sollte daher die am häufigsten verwendete Leseprozedur sein.

PROCEDURE ReadChar(VAR file: File;
VAR ch: CHAR): BOOLEAN;

ReadChar() liest ein einziges Zeichen aus der Datei. Statt *ReadChar()* kann auch die Prozedur *Read()* verwendet werden. *ReadChar()* ist jedoch etwas schneller und daher vorzuziehen, wenn man viele Zeichen einzeln auslesen möchte.

PROCEDURE ReadString(VAR file: File;
VAR to: ARRAY OF CHAR): BOOLEAN;

Liest eine Zeichenkette aus der Datei. Der String wird durch 0X oder ein LineFeed (0AX) abgeschlossen. Maximal werden jedoch LEN(to) Zeichen gelesen.

PROCEDURE ReadBlock(VAR file: File;
to, size: LONGINT): BOOLEAN;

ReadBlock() ist eine sehr niedrige Prozedur zum Einlesen von Datenblocks. Dabei werden *size* Bytes gelesen und in den Speicher ab Adresse *to* geschrieben. *ReadBlock()* sollte nur verwendet werden, wenn dies unumgänglich ist.

PROCEDURE Write(VAR file: File;
VAR from: ARRAY OF BYTE): BOOLEAN;

Schreibt LEN(from) Bytes in die Datei. Dies sollte die am häufigsten verwendete Schreibprozedur sein, da dem Parameter *from* Variablen aller Typen zugewiesen werden können.

PROCEDURE WriteChar(VAR file: File;
ch: CHAR): BOOLEAN;

Schreibt ein Zeichen in die Datei.

PROCEDURE WriteString(VAR file: File;
from: ARRAY OF CHAR): BOOLEAN;

Schreibt die Zeichenkette *from*, gefolgt von einem LineFeed (0AX), in die Datei.

PROCEDURE WriteBlock(VAR file: File;
from, size: LONGINT): BOOLEAN;

Dies ist eine Low-Level-Prozedur, die einen Datenblock in die Datei schreibt. *size* gibt die Größe des Blocks und *from* seine Adresse an. Diese Prozedur sollte nur verwendet werden, wenn dies unumgänglich ist.

PROCEDURE Size(VAR file: File): LONGINT;

Ergibt die Größe der Datei *file* in Bytes.

PROCEDURE Move(VAR file: File;
to: LONGINT): BOOLEAN;

Setzt den Dateizeiger innerhalb der Datei *file*, nach *to*. *to* muß größer oder gleich Null und kleiner oder gleich der Dateigröße sein.

PROCEDURE Forward(VAR file: File;
to: LONGINT): BOOLEAN;
PROCEDURE Backward(VAR file: File;
to: LONGINT): BOOLEAN;

Diese Prozeduren überspringen *to* Bytes in der Datei *file* bzw. springen *to* Bytes zurück, so daß diese Bytes z.B. noch einmal gelesen bzw. geschrieben werden können.

PROCEDURE Delete(VAR file: File): BOOLEAN;

Diese Prozedur sollte aufgerufen werden, wenn beim Erzeugen einer Datei ein Fehler auftrat oder das Programm abgebrochen wurde. Sie schließt die Datei und löscht sie.

PROCEDURE Exists(name: ARRAY OF CHAR): BOOLEAN;

Exists überprüft, ob eine Datei mit dem Namen *name* existiert und gibt TRUE zurück, wenn dies der Fall ist.

Die Fehlercodes:

Nach der Ausführung jeder Prozedur des Moduls FileSystem, mit der Ausnahme von *Exists()*, wird einer der folgenden Fehlercodes in das Feld status der Filevariablen geschrieben:

ok:	Es trat kein Fehler auf.
eof:	Das Dateiende ist erreicht.
readerr:	Es trat ein Lesefehler auf.
writeerr:	Es trat ein Schreibfehler auf.
onlyread:	Es wurde eine Leseprozedur mit einer Datei aufgerufen, die nur zum Schreiben geöffnet wurde.
onlywrite:	Es wurde eine Schreibprozedur mit einer Datei aufgerufen, die nur zum Lesen geöffnet wurde.

toofar:	Mit Move(), Forward() oder Backward() wurde über das Dateiende oder den Dateianfang hinausgesprungen.
outofmem:	Es konnte kein Speicher alloziert werden.
cantopen:	Die Datei konnte nicht geöffnet werden.
cantlock:	Dos.Lock() auf die Datei war nicht möglich.

io:

```
DEFINITION io;

IMPORT d : Dos;

VAR
  out, in : d.FileHandlePtr;
  Me : d.ProcessPtr;

PROCEDURE Write(ch: CHAR);
PROCEDURE WriteLn;
PROCEDURE WriteString(str: ARRAY OF CHAR);
PROCEDURE Tab(n: INTEGER);
PROCEDURE Clear();
PROCEDURE WriteInt(x: LONGINT; n: INTEGER);
PROCEDURE WriteHex(x: LONGINT; n: INTEGER);
PROCEDURE Read(VAR ch: CHAR);
PROCEDURE ReadString(VAR str: ARRAY OF CHAR);
PROCEDURE ReadInt(VAR x: LONGINT): BOOLEAN;
PROCEDURE ReadHex(VAR x: LONGINT): BOOLEAN;
PROCEDURE Length(str: ARRAY OF CHAR): INTEGER;
PROCEDURE Format(VAR str: String; data: LONGINT);

END io.
```

io stellt grundlegende Ein- und Ausgabeprozeduren zur Verfügung. Die Variablen *in* und *out* enthalten die FileHandlePtr der Ein- und Ausgabedateien.

PROCEDURE Write(ch: CHAR);

Gibt ein Zeichen aus.

PROCEDURE WriteLn;

Gibt ein Line-Feed aus, d.h. der Cursor wird an den Anfang der nächsten Zeile gesetzt.

PROCEDURE WriteString(str: ARRAY OF CHAR);

Gibt eine Zeichenkette aus.

PROCEDURE Tab(n: INTEGER);

Gibt einen Tabulatorsprung aus.

PROCEDURE Clear();

Löscht das Ausgabefenster.

PROCEDURE WriteInt(x: LONGINT; n: INTEGER);

Gibt eine n-stellige Integerzahl aus.

PROCEDURE WriteHex(x: LONGINT; n: INTEGER);

Gibt eine n-stellige Hexadezimalzahl aus.

PROCEDURE Read(VAR ch: CHAR);

Liest ein Zeichen von der Tastatur ein.

PROCEDURE ReadString(VAR str: ARRAY OF CHAR);

Liest einen String von der Tastatur ein.

PROCEDURE ReadInt(VAR x: LONGINT): BOOLEAN;

Liest eine Integerzahl ein. Das Ergebnis ist TRUE, wenn eine korrekte Zahl eingegeben wurde.

PROCEDURE ReadHex(VAR x: LONGINT): BOOLEAN;

Liest eine Hexadezimalzahl ein. Das Ergebnis ist TRUE, wenn eine korrekte Zahl eingegeben wurde.

PROCEDURE Length(str: ARRAY OF CHAR): INTEGER;

Bestimmt die Länge einer Zeichenkette. Diese Prozedur ist die gleiche wie Strings.Length. Sie wird hier wiederholt, damit manchmal der Import des Modul Strings vermieden werden kann.

PROCEDURE Format(VAR str: String; data: LONGINT);

Dies ist eine Low-Level Prozedur die in portablen Programmen nicht verwendet werden sollte. Sie ruft Exec.RawDoFmt mit dem String *str* und den Daten an der Adresse *data* auf und gibt das Ergebnis auf den Bildschirm aus.

Lists:

```
DEFINITION Lists;
TYPE
  NodePtr = POINTER TO Node;
  Node = RECORD END;
  List = RECORD END;
  DoProc = PROCEDURE(n: NodePtr);

PROCEDURE Init(VAR list: List);
PROCEDURE AddHead(VAR list: List; n: NodePtr);
PROCEDURE AddTail(VAR list: List; n: NodePtr);
PROCEDURE AddBefore(VAR list: List; n, x: NodePtr);
PROCEDURE AddBehind(VAR list: List; n, x: NodePtr);
PROCEDURE Remove(VAR list: List; n: NodePtr);
PROCEDURE RemHead(VAR list: List): NodePtr;
PROCEDURE RemTail(VAR list: List): NodePtr;
PROCEDURE Empty(VAR list: List): BOOLEAN;
PROCEDURE CountElements(VAR list: List): LONGINT;
PROCEDURE DoForward(VAR list: List; proc: DoProc);
PROCEDURE DoBackward(VAR list: List; proc: DoProc);
PROCEDURE Next(VAR n: NodePtr): BOOLEAN;
PROCEDURE Previous(VAR n: NodePtr): BOOLEAN;
PROCEDURE Head(VAR list: List): NodePtr;
PROCEDURE Tail(VAR list: List): NodePtr;
END Lists.
```

Dieses Modul ermöglicht das einfache Arbeiten mit unsortierten Listen. Die Elemente der Liste sind Erweiterungen des Typs Node und damit zuweisungskompatibel zu den Prozedurparametern.

PROCEDURE Init(VAR list: List);

Init muß aufgerufen werden, bevor mit einer Liste gearbeitet wird. *Init* leert die Liste und bereitet sie für die folgende Benutzung vor.

PROCEDURE AddHead(VAR list: List; n: NodePtr);
PROCEDURE AddTail(VAR list: List; n: NodePtr);
PROCEDURE AddBefore(VAR list: List; n, x: NodePtr);
PROCEDURE AddBehind(VAR list: List; n, x: NodePtr);

Diese Prozeduren fügen das Element *n* in die Liste ein. *AddHead* fügt

es vorne und *AddTail* hinten an. *AddBefore* und *AddBehind* fügen es vor bzw. hinter dem Element *x*, das schon in der Liste sein muß, ein.

PROCEDURE Remove(VAR list: List; n: NodePtr);
PROCEDURE RemHead(VAR list: List): NodePtr;
PROCEDURE RemTail(VAR list: List): NodePtr;

Mit diesen Prozeduren können Elemente aus der Liste entfernt werden. *Remove* entfernt das Element *n*. *RemHead* und *RemTail* entfernen das erste bzw. letzte Element der Liste und geben einen Zeiger darauf zurück, so daß damit noch gearbeitet werden kann.

PROCEDURE Empty(VAR list: List): BOOLEAN;
PROCEDURE CountElements(VAR list: List): LONGINT;

Empty ergibt TRUE, wenn die Liste leer ist. *CountElements* zählt die Anzahl der Listenelemente. *Empty(list)* ist dabei deutlich schneller als *CountElements(list)=0*.

PROCEDURE DoForward(VAR list: List; proc: DoProc);
PROCEDURE DoBackward(VAR list: List; proc: DoProc);

Diese beiden Prozeduren gehen die Liste vorwärts bzw. rückwärts durch und rufen *proc* mit jedem Element der Liste als Parameter auf. Dabei darf *proc* keine Elemente aus der Liste entfernen.

PROCEDURE Next(VAR n: NodePtr): BOOLEAN;
PROCEDURE Previous(VAR n: NodePtr): BOOLEAN;
PROCEDURE Head(VAR list: List): NodePtr;
PROCEDURE Tail(VAR list: List): NodePtr;

Next und Previous bestimmen den Nachfolger bzw. Vorgänger von *n*. Existiert dieser, wird *n* ein Zeiger auf ihn zugewiesen und TRUE zurückgegeben. Sonst wird *n* auf NIL gesetzt. Head und Tail geben einen Zeiger auf das erste bzw. letzte Element der Liste oder NIL bei leerer Liste zurück.

LongRealConversions:

```
DEFINITION LongRealConversions;

PROCEDURE StringToReal(VAR str: ARRAY OF CHAR;
                       VAR r: LONGREAL): BOOLEAN;
PROCEDURE RealToString(r: LONGREAL;
                       VAR str: ARRAY OF CHAR;
                       v, n: INTEGER;
                       exp: BOOLEAN): BOOLEAN;

END LongRealConversions.
```

Diese Prozeduren wandeln Zeichenketten in LONGREAL-Zahlen um und umgekehrt.

PROCEDURE StringToReal(VAR str: ARRAY OF CHAR; VAR r: LONGREAL): BOOLEAN;

Wandelt die Zeichenkette *str* in eine LONGREAL-Zahl um. Die Zeichenkette muß folgender Grammatik entsprechen:

digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9".
sign = ["+"|"-"].
int = {digit}.
real = sign int ["." int] [("E"|"e") sign int].

Der String darf beliebig Leerzeichen enthalten. Sie werden ignoriert. Das Ergebnis der Prozedur ist TRUE, wenn der String eine Zahl enthielt, die der Grammatik entspricht.

Hier einige Beispiele:

String:	LongReal:	Ergebnis:
"+123"	= 123	TRUE
"12.345E2"	= 1234.5	TRUE
""	= 0	TRUE
"-.3E-3"	= -0.0003	TRUE
"12-3"	= ---	FALSE
"23.+4"	= ---	FALSE

PROCEDURE RealToString(r: LONGREAL;
VAR str: ARRAY OF CHAR;
v, n: INTEGER;
exp: BOOLEAN): BOOLEAN;

Wandelt *r* in eine Zeichenkette um. Dabei bekommt das Ergebnis *v* Vorkomma- und *n* Nachkommastellen. Wenn exp TRUE ist, wird ein Exponent erzeugt. *str* muß ohne Exponent mindestens *n*+*v*+3 und mit Exponent mindestens *n*+v+8 Zeichen lang sein. Das Ergebnis ist TRUE, wenn die Zahl in die durch v, *n* und *exp* vorgegebene Schablone paßt.

Beispiel:

RealToString(123.456,str,1,7,TRUE) wandelt 123.456 in die wissenschaftliche Darstellung um. Das Ergebnis ist "1.23456700E+002".

LongRealInOut:

```
DEFINITION LongRealInOut;

PROCEDURE WriteReal(r: LONGREAL;
                    v, n: INTEGER; exp: BOOLEAN): BOOLEAN;
PROCEDURE ReadReal(VAR r: LONGREAL): BOOLEAN;

END LongRealInOut.
```

Die Prozeduren dieses Moduls erlauben die Ein- und Ausgabe von LONGREAL- Zahlen. *WriteReal()* gibt die LONGREAL-Zahl *r* auf dem Bildschirm aus. Die Parameter *v*, *n* und *exp* entsprechen den Parametern der Prozedur LongRealConversions.RealToString.

ReadReal() liest eine LONGREAL-Zahl von der Tastatur ein. Die Syntax entspricht der im Zusammenhang mit LongRealConversions.StringToReal angegebenen Grammatik. Das Ergebnis ist TRUE, wenn eine korrekte Zahl eingegeben wurde.

Mouse:

```
DEFINITION Mouse;

VAR
  leftButton : BOOLEAN;
  rightButton : BOOLEAN;
  midButton : BOOLEAN;

PROCEDURE WaitLMB;
PROCEDURE WaitRMB;

END Mouse.
```

Dieses Modul bietet eine saubere und einfache Methode, die Maustasten abzufragen. Die Variablen *leftButton*, *rightButton* und *midButton* sind jeweils genau dann TRUE, wenn der linke bzw. rechte oder der eventuell existierende mittlere Mouseknopf gedrückt sind.

Um auf die linke oder rechte Maustaste zu warten, kann man *WaitLMB()* oder *WaitRMB()* aufrufen.

NoGuru:

```
DEFINITION NoGuru;

PROCEDURE Assert(cc: BOOLEAN; msg: ARRAY OF CHAR);

END NoGuru.
```

Das Modul NoGuru sollte von jedem guten Amiga-Oberon-Programm, zumindest während der Testphase, importiert werden. Wird dieses Modul importiert, so werden Laufzeitfehler in ein Fehlermeldungen umgewandelt.

Normalerweise führen Laufzeitfehler zum Abbruch von Programmen. Dabei wird keine Meldung ausgegeben, lediglich beim Start vom CLI erscheint die Meldung 'Unable to load "XYZ":', da abgestürzte Oberonprogramme -1 als Return-Wert zurückgeben. Dies ist der einzige Return-Wert, der beim normalen CLI-Start eines Programms eine Meldung ausgibt.

Wird NoGuru importiert, so wird eine detaillierte Fehlermeldung ausgegeben, und das Progamm bleibt noch so lange im Speicher, bis die Return-Taste gedrückt wird. Die Meldung sieht z.B. folgendermaßen aus:

```
Guru #0003: Trap # 3 (Nil-Zeiger dereferenziert)
sr 0004
pc 002AC624
<RETURN>
```

In diesem Fall wurde ein nicht initialisierter Zeiger dereferenziert oder eine nicht initialisierte Prozedurvariable aufgerufen. Dies führt zu einem "TRAP #3". NoGuru übersetzt diesen Trap in die Fehlermeldung 'Nil-Zeiger dereferenziert'. Hinter 'sr' steht hexadezimal der Inhalt des Statusregisters zum Zeitpunkt des Laufzeitfehlers, und hinter 'pc' steht die Adresse, an der das Programm abgestürzt ist.

Wer sich mit 68000er Assembler auskennt, kann mit einem Disassembler den Code in der Nähe des Fehlers disassemblieren und daraus Rückschlüsse auf die Adresse im Quelltext ziehen.

NoGuru gibt folgende Laufzeitfehler aus:

Busfehler
Addressfehler
Illegaler Befehl
Division durch 0
Chk (Bereichsfehler)
TrapV (Überlauf)
Privilegverletzung
Trace-Vektor
Line-A
Line-F
Trap # 0 (Bereichsfehler)
Trap # 1 (ungültiger Case index)
Trap # 2 (Stack überlauf)
Trap # 3 (Nil-Zeiger dereferenziert)
Trap # 4 (Funktion ohne RETURN beendet)
Trap # 5 (Typ-Check Fehler)

Die Prozedur *Assert()* kann verwendet werden, um im Fehlerfall Programme abzubrechen. *Assert()* bricht ein Programm ab und gibt die Fehlermeldung msg aus, wenn die Bedingung cc nicht erfüllt ist.

Beispiel:

```
NEW(p); NoGuru.Assert(p#NIL,"Kein Speicher mehr!");
```

NoGuruRq

```
DEFINITION NoGuruRq;

END NoGuruRq.
```

NoGuruRq entspricht in der Funktionsweise dem Modul NoGuru. Der Unterschied zu NoGuru ist, daß NoGuruRq die Fehlermeldung bei einem Laufzeitfehler in einem Requester und nicht über das Modul io ausgibt. Dadurch wird beim Workbenchsstart kein io-Fenster geöffnet. In dem Requester werden Informationen wie der Inhalt des Statusregisters und der Programmzähler nicht angezeigt.

OberonLib:

```
DEFINITION OberonLib;

VAR
  HaltProc : PROCEDURE();
  OldSP : POINTER TO STRUCT END;
  Result : LONGINT;
  dosCmdLen : LONGINT;
  dosCmdBuf : LONGINT;
  wbStarted : BOOLEAN;
  wbenchMsg : LONGINT;
  Me : POINTER TO STRUCT END;
  MemReqs : LONGSET;

PROCEDURE Mul(a, b: LONGINT): LONGINT;
PROCEDURE ModDiv(a, b: LONGINT): LONGINT;
PROCEDURE New(VAR adr: LONGINT; size: LONGINT);
PROCEDURE Dispose(VAR adr: LONGINT);
PROCEDURE Copy(source: ARRAY OF CHAR; VAR dest: ARRAY OF CHAR);
PROCEDURE StackChk(size: LONGINT);

END OberonLib.
```

OberonLib ist das Basismodul aller Oberon-Programme. Es wird automatisch von jedem anderen Modul importiert. Es hat die Aufgabe, beim Programmstart übergebene Werte, wie die im CLI übergebene Parameterzeile und den zur Verfügung stehenden Stackbereich, zu speichern sowie beim Start von der Workbench aus die Workbench-Message zu holen. Genauere Informationen über Programmstartup-Code stehen in [cbm:lb].

Zudem beinhaltet dieses einige Modul einfache Prozeduren. Der Compiler erzeugt automatisch Aufrufe für diese Prozeduren, wenn man z.B. LONGINT-Werte multipliziert oder Speicher mit NEW() alloziert.

Die Variablen und Prozeduren dieses Moduls sollten mit Vorsicht verwendet und nicht verändert werden. Die Variablen im einzelnen:

HaltProc:

Enthält die Prozedur, zu der die Standardprozedur HALT() springt.

OldSP:

Der Wert des Stackpointers zu Programmbeginn.

Result:

Der Return-Wert des Programms. Der Parameter von HALT() wird in diese Variable geschrieben. HALT(x) kann durch "OberonLib.Result := x; OberonLib.HaltProc;" simuliert werden.

dosCmdLen:

Länge der CLI-Kommandozeile.

dosCmdBuf:

Zeiger auf die CLI-Kommandozeile.

wbStarted:

TRUE, wenn das Programm von der Workbench aus gestartet wurde.

wbenchMsg:

Beim Workbenchstart die Workench-Message.

Me:

Zeiger auf die Task-Struktur des Programms. Exec.FindTask(NIL) ergibt diesen Wert.

MemReqs:

Typ des Speichers, der mit NEW() alloziert wird. Normalerweise ist dies LONGSET{Exec.memClear}. Um z.B. Chip-Memory zu allozieren, sollte Exec.chip gesetzt werden.

Beispiel:

```
INCL(OberonLib.MemReqs,Exec.chip);
NEW(bitplaneptr);
EXCL(OberonLib.MemReqs,Exec.chip);
```

Genauere Informationen zu den verschiedenen Speichertypen können [cmb:exec] entnommen werden.

PROCEDURE Mul(a, b: LONGINT): LONGINT;
PROCEDURE ModDiv(a, b: LONGINT): LONGINT;

Diese beiden Prozeduren werden automatisch aufgerufen, wenn man LONGINT-Zahlen multipliziert, dividiert oder ihren Modulo-Wert berechnet. ModDiv gibt im Register D0 das Ergebnis von "a DIV b" zurück. In D1 steht der Wert von "a MOD b".

PROCEDURE New(VAR adr: LONGINT; size: LONGINT);
PROCEDURE Dispose(VAR adr: LONGINT);

Diese Prozeduren werden zur Speicherverwaltung benötigt. New() alloziert einen Block der Größe *size* und schreibt seine Adresse in *adr*. Dispose gibt diesen Speicher wieder frei. *Dispose()* entspricht exakt der Standardprozedur DISPOSE().

PROCEDURE Copy(source: ARRAY OF CHAR;
VAR dest: ARRAY OF CHAR);

Diese Prozedur wird aufgerufen, wenn die Standardprozedur COPY() verwendet wird.

PROCEDURE StackChk(size: LONGINT);

StackChk wird, bei eingeschalteter Stackprüfung, automatisch zu Beginn jeder Prozedur aufgerufen, um zu prüfen, ob noch *size* Bytes Stapelspeicher vorhanden sind. Ist dies nicht der Fall, wird das Programm abgebrochen.

Random:

```
DEFINITION Random;

PROCEDURE RND(n: INTEGER): INTEGER;
PROCEDURE PutSeed(seed: LONGINT);

END Random.
```

Mit diesem Programm können Pseudo-Zufallszahlen berechnet werden. *RND(n)* ergibt eine Zahl, die zwischen 0 und n-1 liegt. Um zum Beispiel einen Würfel zu simulieren, kann folgende Zeile verwendet werden:

```
augen := RND(6) + 1;
```

Die Zufallszahlen sind bei jedem neuen Programmstart un terschiedlich, da der Zufallsgenerator mit der Videostrahlposition initialisiert wird und damit jedesmal einen anderen Wert enthält.

Werden exakt gleiche Programmläufe benötigt, kann die Ausgangszahl mit *PutSeed()* gesetzt werden. Wird am Programmanfang mit *PutSeed()* immer die gleiche Ausgangszahl gesetzt, liefert auch *RND()* die gleiche Wertefolge.

RealConversions:

```
DEFINITION RealConversions;

PROCEDURE StringToReal(VAR str: ARRAY OF CHAR;
                       VAR  r: REAL): BOOLEAN;
PROCEDURE RealToString(r: REAL;
                       VAR str: ARRAY OF CHAR;
                         v, n: INTEGER;
                          exp: BOOLEAN): BOOLEAN;

END RealConversions.
```

Diese Prozeduren wandeln Zeichenketten in REAL-Zahlen und umgekehrt um.

PROCEDURE StringToReal(VAR str: ARRAY OF CHAR; VAR r: REAL): BOOLEAN;

Wandelt die Zeichenkette *str* in eine REAL-Zahl um. Die Zeichenkette muß folgender Grammatik entsprechen:

digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9".
sign = ["+"|"-"].
int = {digit}.
real = sign int ["." int] [("E"|"e") sign int].

Der String darf beliebig Leerzeichen enthalten. Sie werden ignoriert. Das Ergebnis der Prozedur ist TRUE, wenn der String eine Zahl enthielt, die der Grammatik entspricht.

Hier ein paar Beispiele:

String:	Real:	Ergebnis:
"+123"	= 123	TRUE
"12.345E2"	= 1234.5	TRUE
""	= 0	TRUE
"-.3E-3"	= -0.0003	TRUE
"12-3"	= ---	FALSE
"23.+4"	= ---	FALSE

PROCEDURE RealToString(r: REAL;
VAR str: ARRAY OF CHAR;
v, n: INTEGER;
exp: BOOLEAN): BOOLEAN;

Wandelt *r* in eine Zeichenkette um. Dabei bekommt das Ergebnis *v* Vorkomma- und *n* Nachkommastellen. Wenn exp TRUE ist, wird ein Exponent erzeugt. *str* muß ohne Exponent mindestens *n*+*v*+3 und mit Exponent mindestens *n*+*v*+7 Zeichen lang sein. Das Ergebnis ist TRUE, wenn die Zahl in die durch *v*, *n* und *exp* vorgegebene Schablone paßt.

Beispiel:

RealToString(123.456,str,1,7,TRUE) wandelt 123.456 in die wissenschaftliche Darstellung um. Das Ergebnis ist "1.23456700E+02".

RealInOut:

```
DEFINITION RealInOut;

PROCEDURE WriteReal(r: REAL; v, n: INTEGER; exp: BOOLEAN): BOOLEAN;
PROCEDURE ReadReal(VAR r: REAL): BOOLEAN;

END RealInOut.
```

Die Prozeduren dieses Moduls erlauben die Ein- und Ausgabe von REAL-Zahlen. WriteReal gibt *r* auf den Bildschirm aus. Die Parameter *v*, *n* und *exp* entsprechen den Parametern der Prozedur RealConversions.RealToString.

ReadReal liest entsprechend eine REAL-Zahl von der Tastatur ein.

Requests:

```
DEFINITION Requests;

PROCEDURE Request(head, msg, pos, neg: ARRAY OF CHAR): BOOLEAN;
PROCEDURE Assert(cc: BOOLEAN; msg: ARRAY OF CHAR);
PROCEDURE BreakPoint(msg: ARRAY OF CHAR);

END Requests.
```

Requests erlaubt den einfachen Aufruf von Requestern. Dabei wird Intuitions AutoRequest() benutzt.

PROCEDURE Request(head, msg, pos, neg: ARRAY OF CHAR): BOOLEAN;

Request() öffnet einen Intuition.AutoRequest()-Requester. Dieser besteht aus 2 Textzeilen und 2 Gadgets. *head* und *msg* enthalten die beiden Textzeilen, *pos* und *neg* bestimmen den Inhalt der Gadgets. *pos* kann dabei einen leeren String enthalten ("""). Dies bewirkt, daß der Requester nur 1 Gadget bekommt.

Beispiel:

```
IF Requests.Request("OEd Request:","Save modified text?",
                    "oh, yes","no, thanks") THEN Save(text) END;
```

PROCEDURE Assert(cc: BOOLEAN; msg: ARRAY OF CHAR);

Mit *Assert()* kann ein Programm leicht bei einem Fehler abgebrochen werden. Ist *cc* FALSE, wird ein Requester mit der Meldung *msg* geöffnet und das Programm beendet.

Beispiel:

```
N: .V(p); Requests.Assert(p#NIL,"Kein Speicher mehr!");
```

PROCEDURE BreakPoint(msg: ARRAY OF CHAR);

BreakPoint() kann während der Testphase von Programmen verwendet werden, um Fehler zu lokalisieren. Es öffnet einen Requester mit der Meldung *msg*. Dabei kann man wählen, ob man im Programm fortfahren oder es beenden möchte.

SecureDos:

```
DEFINITION SecureDos;

VAR
  Me : d.ProcessPtr;

PROCEDURE Open(name: LONGINT; accessMode: LONGINT): d.FileHandlePtr;
PROCEDURE Close(file: d.FileHandlePtr);
PROCEDURE Lock(name: LONGINT; accessMode: LONGINT): d.FileLockPtr;
PROCEDURE ParentDir(lock: d.FileLockPtr): d.FileLockPtr;
PROCEDURE DupLock(lock: d.FileLockPtr): d.FileLockPtr;
PROCEDURE UnLock(lock: d.FileLockPtr);

END SecureDos.
```

Die Prozeduren dieses Moduls entsprechen denen der Dos-Library. Der Vorteil ist, daß diese Prozeduren die Resourcen beim Programmende automatisch freigeben, d.h. daß bei einem Programmabbruch Dateien, die noch geöffnet sind, geschlossen werden, und daß alle Locks freigegeben werden.

SecureDos setzt beim Programmende automatisch das 'current directory' auf den Wert vor dem Programmstart.

Strings:

```
DEFINITION Strings;

PROCEDURE Length(str: ARRAY OF CHAR): INTEGER;
PROCEDURE Append(VAR s1: ARRAY OF CHAR;
                 s2: ARRAY OF CHAR);
PROCEDURE Occurs(VAR  s: ARRAY OF CHAR;
                 search: ARRAY OF CHAR): INTEGER;
PROCEDURE OccursPos(VAR  s: ARRAY OF CHAR;
                 search: ARRAY OF CHAR;
                 start: INTEGER     ): INTEGER;
PROCEDURE Cut(VAR  s: ARRAY OF CHAR;
                 from, cnt: INTEGER;
                 VAR to: ARRAY OF CHAR);
PROCEDURE Upper(VAR  s: ARRAY OF CHAR);
PROCEDURE Insert(VAR  s: ARRAY OF CHAR;
                 at: INTEGER;
                 str: ARRAY OF CHAR);
PROCEDURE Delete(VAR  s: ARRAY OF CHAR;
                 at, cnt: INTEGER);
PROCEDURE AppendChar(VAR s: ARRAY OF CHAR;
                 c: CHAR);
PROCEDURE InsertChar(VAR s: ARRAY OF CHAR;
                 at: INTEGER;
                 c: CHAR);

END Strings.
```

Die Prozeduren dieses Moduls dienen zur einfachen Bearbeitung von Zeichenketten.

PROCEDURE Length(str: ARRAY OF CHAR): INTEGER;

Zählt die Anzahl der Zeichen, die *str* enthält, bis zum Ende-Zeichen 0X oder LEN(str).

PROCEDURE Append(VAR s1: ARRAY OF CHAR; s2: ARRAY OF CHAR);

Hängt s2 an s1 an.

Beispiel: s:="Halli"; Strings.Append(s,"Hallo");

Danach enthält s den String "HalliHallo".

PROCEDURE Occurs(VAR s: ARRAY OF CHAR;
search: ARRAY OF CHAR): INTEGER;

Überprüft, ob der String *search* in *s* enthalten ist. Dabei werden Groß- und Kleinbuchstaben unterschieden. Das Ergebnis ist die Position innerhalb von *s*, an der *search* das erstemal vorkommt oder -1, wenn *search* nicht in *s* enthalten ist.

PROCEDURE OccursPos(VAR s: ARRAY OF CHAR;
search: ARRAY OF CHAR;
start: INTEGER): INTEGER;

OccursPos funktioniert ähnlich wie Occurs: Es wird auch geprüft, ob *search* in *s* ab Zeichen Nummer *start* enthalten ist. OccursPos(s1,s2,0) ist also das gleiche wie Occurs(s1,s2). Mit *OccursPos()* können die Positionen von allen Vorkommen von *search* in *s* gefunden werden:

```
p := -1;
LOOP
  p := OccursPos(s,search,p+1);
  IF p<0 THEN EXIT END;
  io.WriteString("Vorkommen an Position ");
  io.WriteInt(p,2); io.WriteLn;
END;
```

PROCEDURE Cut(VAR s: ARRAY OF CHAR;
from, cnt: INTEGER;
VAR to: ARRAY OF CHAR);

Cut nimmt, von Zeichen *from* ausgehend, *cnt* Zeichen aus *s* und kopiert sie nach *to*.

Beispiel: `s1 := "Abstürzende Brieftauben"; Cut(s1,12,5,s2);`

Nun enthält s2 den Teilstring "Brief".

PROCEDURE Upper(VAR s: ARRAY OF CHAR);

Verwandelt alle Kleinbuchstaben von *s* mit der Standardfunktion CAP() in Großbuchstaben.

PROCEDURE Insert(VAR s: ARRAY OF CHAR;
at: INTEGER;
str: ARRAY OF CHAR);

Fügt *str* an der Position *at* in *s* ein.

Beispiel: `s := "Pogo"; Insert(s,2,"go in To");`

Danach enthält *s* den String "Pogo in Togo".

PROCEDURE Delete(VAR s: ARRAY OF CHAR;
at, cnt: INTEGER);

löscht cnt Zeichen von der Position *at* ausgehend aus *s*.

Beispiel: `s := "Pogo in Togo"; Delete(s,2,8);`

Danach enthält *s* den String "Pogo".

PROCEDURE AppendChar(VAR s: ARRAY OF CHAR;
c: CHAR);

Fügt das Zeichen *c* an die Zeichenkette an. Diese Prozedur entspricht *Append()*, sie kann jedoch nur einzelne Zeichen anfügen.

PROCEDURE InsertChar(VAR s: ARRAY OF CHAR;
at: INTEGER;
c: CHAR);

Fügt das Zeichen *c* an der Postion *at* in *s* ein. *InsertChar()* entspricht *Insert()*, kann jedoch nur einzelne Zeichen einfügen.

14. Die Amiga-Interface-Module:

Diese Module ermöglichen den Zugriff auf die Routinen der Amiga-Libraries (Betriebssystembibliotheken) und das Arbeiten mit den Devices (logische Geräte).

ARP:

Dieses Modul ermöglicht den direkten Zugriff auf die Public-Domain Library des 'Amiga Resource Projects'. Dies ist keine Standard-Library, deshalb können Programme, die sie benutzen nur auf Rechnern verwendet werden, die sie installiert haben.

Der Quelltext befindet sich auf der dritten Oberon-Diskette und wird hier nicht wiederholt, da er sehr lang ist.

Audio:

```
MODULE Audio; (* $Implementation- *)

IMPORT e * :  Exec;

CONST
  audioName * = "audio.device";
  hardChannels * = 4;
  allocMinprec * = -128;
  allocMaxprec * = 127;
  free * = e.nonstd+0;
  setPrec * = e.nonstd+1;
  finish * = e.nonstd+2;
  perVol * = e.nonstd+3;
  lock * = e.nonstd+4;
  waitCycle * = e.nonstd+5;
  noUnit * = 32;
  allocate * = noUnit+0;
  pervol * = SHORTSET{4};
  syncCycle * = SHORTSET{5};
  noWait * = SHORTSET{6};
  writeMessage * = SHORTSET{7};
  noAllocation * = -10;
  allocFailed * = -11;
  channelStolen * = -12;

TYPE
  IOAudio * = STRUCT
    request * : e.IORequest;
    allocKey * : INTEGER;
    data * : e.ADDRESS;
    length * : LONGINT;
    period * : INTEGER;
    volume * : INTEGER;
    cycles * : INTEGER;
    writeMsg * : e.Message;
  END;
  IOAudioPtr * = POINTER TO IOAudio;

END Audio.
```

BootBlock:

```
MODULE BootBlock; (* $Implementation- *)

IMPORT sys: SYSTEM;

TYPE
  BootBlock * = STRUCT
    id:ARRAY 4 OF CHAR;
    chkSum * : LONGINT;
    dosBlock * : LONGINT;
  END;

CONST
  bootSects * = 2;
  idDos * = 'DOS';
  idKick * = 'KICK';
  nameDos * = sys.VAL(LONGINT,"DOS"); (* 'DOS' *)
  nameKick * = sys.VAL(LONGINT,"KICK"); (* 'KICK'  *)

END BootBlock.
```

ClipBoard:

```
MODULE Clipboard; (* $Implementation- *)

IMPORT e * :  Exec;

CONST
  clipboardName * = "clipboard.device";
  post * = e.nonstd+0;
  currentReadId * = e.nonstd+1;
  currentWriteId * = e.nonstd+2;
  obsoleteId * = 1;

TYPE
  ClipboardUnitPartial * = STRUCT
    node * : e.Node;
    unitNum * : LONGINT;
  END;
  ClipboardUnitPartialPtr * = POINTER TO ClipboardUnitPartial;
  IOClipboard * = STRUCT
    message * : e.Message;
    device * : e.DevicePtr;
    unit * : e.UnitPtr;
    command * : INTEGER;
    flags * : SHORTSET;
    error * : SHORTINT;
    actual * : LONGINT;
    length * : LONGINT;
    data * : e.ADDRESS;
    offset * : LONGINT;
    clipID * : LONGINT;
  END;
  IOClipboardPtr * = POINTER TO IOClipboardPtr;

CONST
  primaryClip * = 0;

TYPE
  SatisfyMsg * = STRUCT
    msg * : e.Message;
    unit * : INTEGER;
    clipID * : LONGINT;
  END;
  SatisfyMsgPtr * = POINTER TO SatisfyMsg;

END Clipboard.
```

Console:

```
MODULE Console;

IMPORT e: Exec,
       ie: InputEvent,
       km: KeyMap;

CONST
  consoleName * = "console.device";
  askKeyMap * = e.nonstd+0;
  setKeyMap * = e.nonstd+1;
  askDefaultKeyMap * = e.nonstd+2;
  setDefaultKeyMap * = e.nonstd+3;
  primary * = 0;
  bold * = 1;
  italic * = 3;
  underscore * = 4;
  negative * = 7;
  black * = 30;
  red * = 31;
  green * = 32;
  yellow * = 33;
  blue * = 34;
  magenta * = 35;
  cyan * = 36;
  white * = 37;
  default * = 39;
  blackBg * = 40;
  redBg * = 41;
  greenBg * = 42;
  yellowBg * = 43;
  blueBg * = 44;
  magentaBg * = 45;
  cyanBg * = 46;
  whiteBg * = 47;
  defaultBg * = 49;
  clr0 * = 30;
  clr1 * = 31;
  clr2 * = 32;
  clr3 * = 33;
  clr4 * = 34;
  clr5 * = 35;
  clr6 * = 36;
  clr7 * = 37;
  clr0Bg * = 40;
  clr1Bg * = 41;
```

```
  clr2Bg * = 42;
  clr3Bg * = 43;
  clr4Bg * = 44;
  clr5Bg * = 45;
  clr6Bg * = 46;
  clr7Bg * = 47;
  dsrCpr * = 6;
  ctcHSetTab * = 0;
  ctcHClrTab * = 2;
  ctcHClrTabsAll * = 5;
  tbcHClrTab * = 0;
  tbcHClrTabsAll * = 3;
  mLnm * = 20;
  mAsm * = ">1";
  mAwm * = "?7";

VAR
  condev* : e.DevicePtr; (* Dies muß auf das Console-Device zeigen*)

PROCEDURE CDInputHandler* {condev,-42}(events{8}:ie.InputEventPtr;
                                       device1{9}:e.ADDRESS):ie.InputEventPtr;
PROCEDURE RawKeyConvert* {condev,-48}(events{8}:ie.InputEventPtr;
                                      buffer{9}:e.ADDRESS;
                                      length{1}:LONGINT;
                                      keyMap{10}:km.KeyMapPtr):LONGINT;

END Console.
```

DiskFont:

```
MODULE DiskFont;

IMPORT e: Exec,
       g: Graphics,
       I: Intuition,
       s: SYSTEM;

CONST
  maxFontPath * = 256;
  maxFontName * = 32;
  fchId * = 0F00H;
  dfhId * = 0F80H;

(* AvailFontTypes *)
  memory * = 0;
  disk * = 1;

TYPE
  FontContents * = STRUCT
    fileName * : ARRAY maxFontPath OF CHAR;
    ySize * : INTEGER;
    style * : SHORTSET; (* g.FontStyles *)
    flags * : SHORTSET; (* g.FontFlags *)
  END;
  FontContentsHeader * = STRUCT
    fileId * : INTEGER;
    numEntries * : INTEGER;
(*fc * : ARRAY numEntries OF FontContents *)
  END;
  FontContentsHeaderPtr * = POINTER TO FontContentsHeader;
  DiskFontHeader * = STRUCT
    df * : e.Node;
    fileId * : INTEGER;
    revision * : INTEGER;
    segment * : e.BPTR;
    name * : ARRAY maxFontName OF CHAR;
    tf * : g.TextFont
  END;
  AvailFont * = STRUCT
    type * : SET; (* AvailFontTypes *)
    attr * : g.TextAttr;
  END;
  AvailFontHeader * = STRUCT
    numEntries * : INTEGER;
(*af * : ARRAY numEntries OF AvailFont;*)
```

```
  END;
  AvailFontHeaderPtr * = POINTER TO AvailFontHeader;

VAR
  base * : e.LibraryPtr;

PROCEDURE AvailFonts * {base,-36}(
                    buffer{8}: e.ADDRESS;
                    bufBytes{0}: LONGINT;
                    types{1}: SET (* AvailFontTypes *)):LONGINT;
PROCEDURE OpenDiskFont * {base,-30}(
                    textAttr{8}: g.TextAttrPtr): g.TextFontPtr;

(* $OvflChk- $RangeChk- $StackChk- $NilChk- $ReturnChk- $CaseChk- *)

BEGIN

  base := e.OpenLibrary("diskfont.library",33);
  IF base=NIL THEN
    s.SETREG(0,l.DisplayAlert(0,"x00x64x14missing diskfont.library!",50));
    HALT(0)
  END;

CLOSE

  IF base#NIL THEN e.CloseLibrary(base) END;

END DiskFont.
```

Dos:

```
MODULE Dos;

IMPORT e: Exec, s: SYSTEM;

CONST
 dosName * = "dos.library";

(* accessMode bei Open(): *)
 readWrite * = 1004;
 readOnly * = 1005;
 oldFile * = readOnly;
 newFile * = 1006;

(* mode der Seek(): *)
 beginning * = -1;
 current * = 0;
 end * = 1;

(* accessMode bei Lock(): *)
 sharedLock * = -2;
 exclusiveLock * = -1;
 accessRead * = sharedLock;    (* synonym *)
 accessWrite * = exclusiveLock; (* synonym *)
 ticksPerSecond * = 50;

TYPE
 BSTR * = BPOINTER TO ARRAY 256 OF CHAR;
 DeviceListPtr * = BPOINTER TO DeviceList;
 DeviceListVolPtr * = BPOINTER TO DeviceListVol;
 FileHandlePtr * = BPOINTER TO FileHandle;
 FileLockPtr * = BPOINTER TO FileLock;
 Date * = STRUCT
  days * : LONGINT;
  minute * : LONGINT;
  tick * : LONGINT;
 END;
 DatePtr * = POINTER TO Date;

CONST
(* ProtectionFlags *)
  delete     * = 0;
  execute    * = 1;
  writeProt  * = 2;
  readProt   * = 3;
  archive    * = 4;
```

```
  pure      * = 5;
  script    * = 6;
  hidden    * = 7;

TYPE
 FileInfoBlockPtr * = POINTER TO FileInfoBlock;
 FileInfoBlock * = STRUCT
  diskKey * : LONGINT;
  dirEntryType * : LONGINT;
  fileName * : ARRAY 108 OF CHAR;
  protection * : LONGSET; (* ProtectionFlags *)
  entryType * : LONGINT;
  size * : LONGINT;
  numBlocks * : LONGINT;
  date * : Date;
  comment * : ARRAY 80 OF CHAR;
  reserved * : ARRAY 36 OF CHAR;
 END;
 InfoDataPtr * = POINTER TO InfoData;
 InfoData * = STRUCT
  numSoftErrors * : LONGINT;
  unitNumber * : LONGINT;
  diskState * : LONGINT;
  numBlocks * : LONGINT;
  numBlocksUsed * : LONGINT;
  bytesPerBlock * : LONGINT;
  diskType * : LONGINT;
  volumeNode * : DeviceListPtr;
  inUse * : LONGINT;
 END;

CONST
(* InfoData.diskState * : *)
 writeProtect * = 80;
 validating * = 81;
 validated * = 82;

(* InfoData.diskType * : *)
 noDiskPresent  * = -1;
 unreadableDisk * = s.VAL(LONGINT,"BAD"); (* 42414400H *)
 dosDisk        * = s.VAL(LONGINT,"DOS"); (* 444F5300H *)
 notReallyDos   * = s.VAL(LONGINT,"NDOS" ); (* 4E444F53H *)
 kickstartDisk  * = s.VAL(LONGINT,"KICK" ); (* 4B49434BH *)

(* Ergebnisse von IoErr() * : *)
 noFreeStore * = 103;
 taskTableFull * = 105;
 lineTooLong * = 120;
```

```
fileNotObject * = 121;
invalidResidentLibrary * = 122;
noDefaultDir * = 201;
objectInUse * = 202;
objectExists * = 203;
dirNotFound * = 204;
objectNotFound * = 205;
badStreamName * = 206;
objectTooLarge * = 207;
actionNotKnown * = 209;
invalidComponentName * = 210;
invalidLock * = 211;
objectWrongType * = 212;
diskNotValidated * = 213;
diskWriteProtected * = 214;
renameAcrossDevices * = 215;
directoryNotEmpty * = 216;
tooManyLevels * = 217;
deviceNotMounted * = 218;
seekError * = 219;
commentTooBig * = 220;
diskFull * = 221;
deleteProtected * = 222;
writeProtected * = 223;
readProtected * = 224;
notADosDisk * = 225;
noDisk * = 226;
noMoreEntries * = 232;

(* ReturnCode für Exit() * :  (in Oberon bitte stattdessen HALT() nehmen!) *)
ok * = 0;
warn * = 5;
error * = 10;
fail * = 20;

(* break Signalbits *)
ctrlC * = 12;
ctrlD * = 13;
ctrlE * = 14;
ctrlF * = 15;

TYPE
CommandLineInterfacePtr * = BPOINTER TO CommandLineInterface;
ProcessId * = e.MsgPortPtr;
ProcessPtr * = POINTER TO Process;
Process * = STRUCT
 task * : e.Task;
 msgPort * : e.MsgPort;
```

```
 pad * : INTEGER;
 segList * : e.BPTR;
 stackSize * : LONGINT;
 globVec * : e.ADDRESS;
 taskNum * : LONGINT;
 stackBase * : e.BPTR;
 result2 * : LONGINT;
 currentDir * : FileLockPtr;
 cis * : FileHandlePtr;
 cos * : FileHandlePtr;
 consoleTask * : ProcessId;
 fileSystemTask * : ProcessId;
 cli * : CommandLineInterfacePtr;
 returnAddr * : e.ADDRESS;
 pktWait * : e.ADDRESS;
 windowPtr * : e.ADDRESS;
END;
FileHandle * = STRUCT
 link * : e.MessagePtr;
 port * : e.MsgPortPtr;
 type * : ProcessId;
 buf * : LONGINT;
 pos * : LONGINT;
 end * : LONGINT;
 func1 * : LONGINT;
 func2 * : LONGINT;
 func3 * : LONGINT;
 arg1 * : LONGINT;
 arg2 * : LONGINT;
END;
DosPacket * = STRUCT
 link * : e.MessagePtr;
 port * : e.MsgPortPtr;
 type * : LONGINT; (* Action *)
 res1 * : LONGINT; (* Status *)
 res2 * : LONGINT; (* Status2 *)
 arg1 * : LONGINT; (* BufAddr *)
 arg2 * : LONGINT;
 arg3 * : LONGINT;
 arg4 * : LONGINT;
 arg5 * : LONGINT;
 arg6 * : LONGINT;
 arg7 * : LONGINT;
END;
DosPacketPtr * = POINTER TO DosPacket;
StandardPacket * = STRUCT
 msg * : e.Message;
 pkt * : DosPacket;
```

```
 END;
 StandardPacketPtr * = POINTER TO StandardPacket;

CONST

(* DosPacket.type *)
 nil * = 0;
 getBlock * = 2;
 setMap * = 4;
 die * = 5;
 event * = 6;
 currentVolume * = 7;
 locateObject * = 8;
 renameDisk * = 9;
 write * = ORD('W');
 read * = ORD('R');
 freeLock * = 15;
 deleteObject * = 16;
 renameObject * = 17;
 moreCache * = 18;
 copyDir * = 19;
 waitChar * = 20;
 setProtect * = 21;
 createDir * = 22;
 examineObject * = 23;
 examineNext * = 24;
 diskInfo * = 25;
 info * = 26;
 flush * = 27;
 setComment * = 28;
 parent * = 29;
 timer * = 30;
 inhibit * = 31;
 diskType * = 32;
 diskChange * = 33;
 setDate * = 34;
 screenMode * = 994;
 readReturn * = 1001;
 writeReturn * = 1002;
 findUpdate * = 1004;
 findInput * = 1005;
 findOutput * = 1006;
 actionEnd * = 1007;
 seek * = 1008;
 truncate * = 1022;
 writeLock * = 1023;

TYPE
```

```
RootNodePtr * = POINTER TO RootNode;
DosLibrary * = STRUCT
 lib * : e.Library;
 root * : RootNodePtr;
 gv * : e.ADDRESS;
 a2 * : LONGINT;
 a5 * : LONGINT;
 a6 * : LONGINT;
END;
DosLibraryPtr * = POINTER TO DosLibrary;
TaskArray * = STRUCT
 maxCli * : LONGINT;
 cli * : ARRAY 100 OF ProcessId;
END;
DosEnvecPtr * = BPOINTER TO DosEnvec;
DosInfoPtr * = BPOINTER TO DosInfo;
FileSysStartupMsgPtr * = BPOINTER TO FileSysStartupMsg;
TaskArrayPtr * = BPOINTER TO TaskArray;
ResidentSegmentPtr * = BPOINTER TO ResidentSegment;
ResidentSegment * = STRUCT
 next * : ResidentSegmentPtr;
 usecount * : LONGINT;
 segment * : e.BPTR;
(* name * : ARRAY OF CHAR; [0] = length, [1..] = string *)
END;
RootNode * = STRUCT
 taskArray * : TaskArrayPtr;
 consoleSegment * : e.BPTR;
 time * : Date;
 restartSeg * : e.BPTR;
 info * : DosInfoPtr;
 fileHandlerSegment * : e.BPTR;
END;
DosInfo * = STRUCT
 mcName * : BSTR;
 devInfo * : DeviceListPtr;
 devices * : e.BPTR;
 handlers * : e.BPTR;
 netHand * : ResidentSegmentPtr;
END;
PathInfoPtr * = BPOINTER TO PathInfo;
PathInfo * = STRUCT
 nextPath * : PathInfoPtr;
 lock * : FileLockPtr;
END;
CommandLineInterface * = STRUCT
 result2 * : LONGINT;
 setName * : BSTR;
```

```
  commandDir * : PathInfoPtr;
  returnCode * : LONGINT;
  commandName * : BSTR;
  failLevel * : LONGINT;
  prompt * : BSTR;
  standardInput * : FileHandlePtr;
  currentInput * : FileHandlePtr;
  commandFile * : BSTR;
  interactive * : LONGINT; (* LONGBOOLEAN *)
  background * : LONGINT; (* LONGBOOLEAN *)
  currentOutput * : FileHandlePtr;
  defaultStack * : LONGINT;
  standardOutput * : FileHandlePtr;
  module * : e.BPTR;
 END;

CONST
(* DeviceListType *)
 device    * = 0;
 directory * = 1;
 volume    * = 2;

TYPE
 DeviceList * = STRUCT
  next * : DeviceListPtr;
  type * : LONGINT; (* DeviceListType *)
  task * : ProcessId;
  lock * : FileLockPtr;
  handler * : BSTR;
  stackSize * : LONGINT;
  priority * : LONGINT;
  startup * : FileSysStartupMsgPtr;
  segList * : e.BPTR;
  globVec * : e.BPTR;
  name * : BSTR;
 END;
 DeviceListVol * = STRUCT  (* für Volumes *)
  next * : DeviceListPtr;
  type * : LONGINT; (* DeviceListType *)
  task * : ProcessId;
  lock * : FileLockPtr;
  volumeDate * : Date;
  lockList * : FileLockPtr;
  diskType * : LONGINT;
  unused * : LONGINT;
  name * : BSTR;
 END;
```

```
DeviceNode * = DeviceList;
DeviceNodePtr * = POINTER TO DeviceNode;
DeviceNodeVol * = DeviceListVol;
DeviceNodeVolPtr * = POINTER TO DeviceNodeVol;

DosEnvec * = STRUCT
 tableSize * : LONGINT;
 sizeBlock * : LONGINT;
 secOrg * : LONGINT;
 surfaces * : LONGINT;
 sectorsPerBlock * : LONGINT;
 blocksPerTrack * : LONGINT;
 reserved * : LONGINT;
 preAlloc * : LONGINT;
 interleave * : LONGINT;
 lowCyl *, highCyl : LONGINT;
 numBuffers * : LONGINT;
 bufMemType * : LONGINT;
 maxTransfers * : LONGINT;
 mask * : LONGSET;
 bootPri * : LONGINT;
 dosType * : ARRAY 4 OF CHAR;
END;
FileLock * = STRUCT
 link * : FileLockPtr;
 key * : LONGINT;
 access * : LONGINT;
 task * : ProcessId;
 volume * : DeviceListPtr;
END;
FileSysStartupMsg * = STRUCT
 unit * : LONGINT;
 device * : BSTR;
 environ * : DosEnvecPtr;
 flags * : LONGSET;
END;

VAR dos * :  DosLibraryPtr;

PROCEDURE Close* {dos,- 36}(file{1}:  FileHandlePtr);
PROCEDURE CreateDir* {dos,-120}(name{1}: ARRAY OF CHAR): FileLockPtr;
PROCEDURE CreateProc* {dos,-138}(name{1}: ARRAY OF CHAR;
                                 pri{2}: LONGINT; segment{3}: e.BPTR;
                                 stackSize{4}: LONGINT): ProcessId;
PROCEDURE CurrentDir* {dos,-126}(lock{1}: FileLockPtr): FileLockPtr;
PROCEDURE DateStamp* {dos,-192}(VAR v{1}: Date);
PROCEDURE Delay* {dos,-198}(ticks{1}: LONGINT);
PROCEDURE DeleteFile* {dos,- 72}(name{1}: ARRAY OF CHAR): BOOLEAN;
```

```
PROCEDURE DeviceProc* {dos,-174}(name{1}: ARRAY OF CHAR): ProcessId;
PROCEDURE DupLock* {dos,- 96}(lock{1}: FileLockPtr): FileLockPtr;
PROCEDURE Examine* {dos,-102}(lock{1}: FileLockPtr;
        infoBlock{2}: FileInfoBlockPtr): BOOLEAN;
PROCEDURE Execute* {dos,-222}(commandString{1}: ARRAY OF CHAR;
        input{2}: FileHandlePtr;
        output{3}: FileHandlePtr): BOOLEAN;
PROCEDURE Exit* {dos,-144}(returnCode{1}: LONGINT);
PROCEDURE ExNext* {dos,-108}(lock{1}: FileLockPtr;
        infoBlock{2}: FileInfoBlockPtr): BOOLEAN;
PROCEDURE GetPacket* {dos,-162}(wait{1}: LONGINT): DosPacketPtr;
PROCEDURE Info* {dos,-114}(lock{1}: FileLockPtr;
        parameterBlock{2}: InfoDataPtr): BOOLEAN;
PROCEDURE Input* {dos,- 54}(): FileHandlePtr;
PROCEDURE IoErr* {dos,-132}(): LONGINT;
PROCEDURE IsInteractive* {dos,-216}(file{1}: FileHandlePtr): BOOLEAN;
PROCEDURE LoadSeg* {dos,-150}(name{1}: ARRAY OF CHAR): e.BPTR;
PROCEDURE Lock* {dos,- 84}(name{1}: ARRAY OF CHAR;
            accessMode{2}: LONGINT): FileLockPtr;
PROCEDURE Open* {dos,- 30}(name{1}: ARRAY OF CHAR;
        accessMode{2}: LONGINT): FileHandlePtr;
PROCEDURE Output* {dos,- 60}(): FileHandlePtr;
PROCEDURE ParentDir* {dos,-210}(lock{1}: FileLockPtr): FileLockPtr;
PROCEDURE QueuePacket* {dos,-168}(packet{1}: DosPacketPtr): LONGINT;
PROCEDURE Read* {dos,- 42}(file{1}: FileHandlePtr;
            buffer{2}: ARRAY OF BYTE;
            length{3}: LONGINT): LONGINT;
PROCEDURE Rename* {dos,- 78}(oldName{1},newName{2}: ARRAY OF CHAR):
BOOLEAN;
PROCEDURE Seek* {dos,- 66}(file{1}: FileHandlePtr;
            position{2}: LONGINT;
            mode{3}: LONGINT): LONGINT;
PROCEDURE SetComment* {dos,-180}(
             name{1},comment{2}: ARRAY OF CHAR): BOOLEAN;
PROCEDURE SetProtection* {dos,-186}(name{1}: ARRAY OF CHAR;
                            mask{2}: LONGSET (* ProtectionFlags *)
                            ): BOOLEAN;
PROCEDURE UnLoadSeg* {dos,-156}(segment{1}: e.BPTR);
PROCEDURE UnLock* {dos,- 90}(lock{1}: FileLockPtr);
PROCEDURE WaitForChar* {dos,-204}(file{1}: FileHandlePtr;
                           timeout{2}: LONGINT): BOOLEAN;
PROCEDURE Write* {dos,- 48}(file{1}: FileHandlePtr;
                        buffer{2}: ARRAY OF BYTE;
                        length{3}: LONGINT): LONGINT;

(* $OvflChk- $RangeChk- $StackChk- $NilChk- $ReturnChk- $CaseChk- *)
```

```
BEGIN

 dos := s.VAL(DosLibraryPtr,e.OpenLibrary(dosName,33));
 IF dos = NIL THEN HALT(0) END;

CLOSE

 IF dos#NIL THEN e.CloseLibrary(s.VAL(e.LibraryPtr,dos)) END;

END Dos.
```

Exec:

```
MODULE Exec; (* $Implementation- *)

TYPE
  ADDRESS * = LONGINT;
  BPTR * = BPOINTER TO LONGINT;
  PROC * = PROCEDURE;

CONST
(* IntFlags *)
  tbeInt      * = 0;
  dskblk      * = 1;
  ifSoftint   * = 2;
  ports       * = 3;
  coper       * = 4;
  vertb       * = 5;
  blit        * = 6;
  aud0i       * = 7;
  aud1i       * = 8;
  aud2i       * = 9;
  aud3i       * = 10;
  rbfInt      * = 11;
  disksync    * = 12;
  exter       * = 13;
  inten       * = 14;
  intSet      * = 15;
  IntFlagsMax * = 16;

(* NodeType *)
  unknown     * = 0;
  task        * = 1;
  interrupt   * = 2;
  device      * = 3;
  msgPort     * = 4;
  message     * = 5;
  freeMsg     * = 6;
  replyMsg    * = 7;
  resource    * = 8;
  library     * = 9;
  memory      * = 10;
  softInt     * = 11;
  font        * = 12;
  process     * = 13;
  semaphore   * = 14;
  signalSem   * = 15;
  bootNode    * = 16;
```

```
TYPE
  NodePtr * = POINTER TO Node;
  Node * = STRUCT
    succ * : NodePtr;
    pred * : NodePtr;
    type * : SHORTINT; (* NodeType *)
    pri * : SHORTINT;
    name * : LONGINT;
  END;
  MinNodePtr * = POINTER TO MinNode;
  MinNode * = STRUCT
    succ * : MinNodePtr;
    pred * : MinNodePtr;
  END;
  List * = STRUCT
    head * : NodePtr;
    tail * : NodePtr;
    tailPred * : NodePtr;
    type * : SHORTINT; (* NodeType *)
    pad * : BYTE;
  END;
  ListPtr * = POINTER TO List;
  MinList * = STRUCT
    head * : MinNodePtr;
    tail * : MinNodePtr;
    tailPred * : MinNodePtr;
  END;
  MinListPtr * = POINTER TO MinList;
  Interrupt * = STRUCT
    node * : Node;
    data * : LONGINT;
    code * : PROC;
  END;
  InterruptPtr * = POINTER TO Interrupt;
  IntVector * = STRUCT
    data * : LONGINT;
    code * : PROC;
    node * : NodePtr;
  END;
  SoftIntList * = STRUCT
    list * : List;
    pad * : INTEGER;
  END;

CONST
(* MemReqs *)
  public   * = 0;
  chip     * = 1;
```

```
  fast     * = 2;
  memClear * = 16;
  largest  * = 17;

TYPE
  MemChunkPtr * = POINTER TO MemChunk;
  MemChunk * = STRUCT
    next * : MemChunkPtr;
    bytes * : LONGINT;
  END;
  MemHeader * = STRUCT
    node * : Node;
    attributes * : SET; (* MemReqs *)
    first * : MemChunkPtr;
    lower * : LONGINT;
    upper * : LONGINT;
    free * : LONGINT;
  END;
  MemHeaderPtr * = POINTER TO MemHeader;
  MemEntry * = STRUCT
    addr * : LONGINT; (* or * : reqs * : LONGSET; (* MemReqs *) *)
    length * : LONGINT;
  END;
  MemList * = STRUCT
    node * : Node;
    numEntries * : INTEGER;
(*me * : ARRAY munEntries OF MemEntry;*)
  END;
  MemListPtr * = POINTER TO MemList;

CONST
(* MsgPortAction *)
  signal * = 0;
  softint * = 1;
  ignore * = 2;

TYPE
  TaskPtr * = POINTER TO Task;
  MsgPort * = STRUCT
    node * : Node;
    flags * : SHORTINT; (* MsgPortAction *)
    sigBit * : SHORTINT;
    sigTask * : TaskPtr;
    msgList * : List;
    END;
  MsgPortSoftInt * = STRUCT  (* use when MsgPort.flags * = softint *)
    node * : Node;
    flags * : SHORTINT; (* MsgPortAction *)
```

```
    pad0 * : BYTE;
    softInt * : InterruptPtr;
    msgList * : List;
  END;
  MsgPortPtr * = POINTER TO MsgPort;
  Message * = STRUCT
    node * : Node;
    replyPort * : MsgPortPtr;
    length * : INTEGER;
  END;
  MessagePtr * = POINTER TO Message;

CONST
(* TaskFlags *)
  procTime   * = 0;
  stackChk   * = 4;
  exception  * = 5;
  switch     * = 6;
  launch     * = 7;

(* TaskState *)
  inval   * = 0;
  added   * = 1;
  run     * = 2;
  ready   * = 3;
  wait    * = 4;
  except  * = 5;
  removed * = 6;

TYPE
  Task * = STRUCT
    node * : Node;
    flags * : SHORTSET; (* TaskFlags *)
    state * : SHORTSET; (* TaskState *)
    idNestCnt * : SHORTINT;
    tdNestCnt * : SHORTINT;
    sigAlloc * : LONGSET;
    sigWait * : LONGSET;
    sigRecvd * : LONGSET;
    sigExcept * : LONGSET;
    trapAlloc * : SET;
    trapAble * : SET;
    exceptData * : LONGINT;
    exceptCode * : PROC;
    trapData * : LONGINT;
    trapCode * : PROC;
    spReg * : LONGINT;
    spLower * : LONGINT;
```

```
    spUpper * : LONGINT;
    switch * : PROC;
    launch * : PROC;
    memEntry * : List;
    userData * : LONGINT;
  END;

CONST
(* Vordefinierte Signalnummern *)
  sigAbort * = 0;
  sigChild * = 1;
  sigBlit * = 4;
  sigSingle * = 4;
  sigDos * = 8;

  vectSize * = 6;
  reserved * = 4;
  base * = -vectSize;
  userDef * = base-reserved*vectSize;
  nonStd * = userDef;
  extFunc * = -24;
  expunge * = -18;
  close * = -12;
  open * = -6;

(* LibFlags *)
  summing * = 0;
  changed * = 1;
  sumUsed * = 2;
  delExp  * = 3;

TYPE
  Library * = STRUCT
    node * : Node;
    flags * : SHORTSET; (* LibFlags *);
    pad * : BYTE;
    negSize * : INTEGER;
    posSize * : INTEGER;
    version * : INTEGER;
    revision * : INTEGER;
    idString * : LONGINT;
    sum * : LONGINT;
    openCnt * : INTEGER;
  END;
  LibraryPtr * = POINTER TO Library;
  Device * = STRUCT
    library * : Library;
  END;
```

```
  DevicePtr * = POINTER TO Device;

CONST
(* UnitFlags *)
  active * = 0;
  inTask * = 1;.

TYPE

  UnitPtr * = POINTER TO Unit;
  Unit * = STRUCT
    msgPort * : MsgPort;
    flags * : SHORTSET; (* UnitFlags *)
    pad * : BYTE;
    openCnt * : INTEGER;
  END;

CONST
    (* Standardbefehle für IORequest.command *)
  invalid * = 0;
  reset * = 1;
  read * = 2;
  write * = 3;
  update * = 4;
  clear * = 5;
  stop * = 6;
  start * = 7;
  flush * = 8;
  nonstd * = 9;
    (* Standard Fehlerwerte fuer IORequest.error *)
  openFail * = -1;
  aborted * = -2;
  noCmd * = -3;
  badLength * = -4;
    (* Offstes der Devicespezifischen Funktionen  *)
  abortIO * = -36;
  beginIO * = -30;

  quick * = 0;

TYPE
  IORequest * = STRUCT
    message * : Message;
    device * : DevicePtr;
    unit * : UnitPtr;
    command * : INTEGER;
    flags * : SHORTSET;
    error * : SHORTINT;
```

```
  END;
  IORequestPtr * = POINTER TO IORequest;
  IOStdReq * = STRUCT
    message * : Message;
    device * : DevicePtr;
    unit * : UnitPtr;
    command * : INTEGER;
    flags * : SHORTSET;
    error * : SHORTINT;
    actual * : LONGINT;
    length * : LONGINT;
    data * : LONGINT;
    offset * : LONGINT;
  END;
  IOStdReqPtr * = POINTER TO IOStdReq;
  Semaphore * = STRUCT
    msgPort * : MsgPort;
    bids * : INTEGER;
  END;
  SemaphorePtr * = POINTER TO Semaphore;
  SemaphoreRequest * = STRUCT
    link * : MinNode;
    waiter * : TaskPtr;
  END;
  SignalSemaphore * = STRUCT
    link * : Node;
    nestCount * : INTEGER;
    waitQueue * : MinList;
    multipleLink * : SemaphoreRequest;
    owner * : TaskPtr;
    queueCount * : INTEGER;
  END;
  SignalSemaphorePtr * = POINTER TO SignalSemaphore;

CONST
(* ResidentFlags *)
  coldstart * = 0;
  autoinit  * = 7;

TYPE
  ResidentPtr * = POINTER TO Resident;
  Resident * = STRUCT
    matchWord * : INTEGER;
    matchTag * : ResidentPtr;
    endSkip * : LONGINT;
    flags * : SHORTSET; (* ResidentFlags *)
    version * : BYTE;
    type * : SHORTINT; (* NodeType *)
```

```
    pri * : SHORTINT;
    name * : LONGINT;
    idString * : LONGINT;
    init * : LONGINT;
  END;

CONST
(* add to Alertnumber for Deadend-Alerts: *)
  deadEnd * = MIN(LONGINT);

CONST
  matchword * = 04AFCH;

(* AttnFlags *)
    m68010 * = 0;
    m68020 * = 1;
    m68881 * = 4;

TYPE
  ExecBase * = STRUCT
    libNode * : Library;
    softVer * : INTEGER;
    lowMemChkSum * : INTEGER;
    chkBase * : LONGINT;
    coldCapture * : LONGINT;
    coolCapture * : LONGINT;
    warmCapture * : LONGINT;
    sysStkUpper * : LONGINT;
    sysStkLower * : LONGINT;
    maxLocMem * : LONGINT;
    debugEntry * : LONGINT;
    debugData * : LONGINT;
    alertData * : LONGINT;
    maxExtMem * : LONGINT;
    chkSum * : INTEGER;
    intVects * : ARRAY IntFlagsMax OF IntVector;
    thisTask * : TaskPtr;
    idleCount * : LONGINT;
    dispCount * : LONGINT;
    quantum * : INTEGER;
    elapsed * : INTEGER;
    sysFlags * : INTEGER;
    idNestCnt * : SHORTINT;
    tdNestCnt * : SHORTINT;
    attnFlags * : SET; (* AttnFlags *);
    attnResched * : INTEGER;
    resModules * : LONGINT;
    taskTrapCode * : PROC;
```

```
    taskExceptCode * : PROC;
    taskExitCode * : PROC;
    taskSigAlloc * : LONGSET;
    taskTrapAlloc * : SET;
    memList * : List;
    resourceList * : List;
    deviceList * : List;
    intrList * : List;
    libList * : List;
    portList * : List;
    taskReady * : List;
    taskWait * : List;
    softInts * : ARRAY 5 OF SoftIntList;
    lastAlert * : ARRAY 4 OF LONGINT;
    vBlankFrequency * : BYTE;
    powerSupplyFrequency * : BYTE;
    semaphoreList * : List;
    kickMemPtr * : LONGINT;
    kickTagPtr * : LONGINT;
    kickCheckSum * : LONGINT;
    execBaseReserved * : ARRAY 10 OF BYTE;
    execBaseNewReserved * : ARRAY 20 OF BYTE;
  END;
  ExecBasePtr * = POINTER TO ExecBase;

VAR
  exec* [4] : ExecBasePtr;

PROCEDURE AbortIO* {exec,-480}(iORequest{9}: ADDRESS);
PROCEDURE AddDevice* {exec,-432}(device{9}: DevicePtr);
PROCEDURE AddHead* {exec,-240}(list{8}: ListPtr;
                               node{9}: ADDRESS);
PROCEDURE AddIntServer* {exec,-168}(intNum{0}: LONGINT;
                                    interrupt{9}: InterruptPtr);
PROCEDURE AddLibrary* {exec,-396}(library{9}: LibraryPtr);
PROCEDURE AddMemList* {exec,-618}(size{0}: LONGINT;
                                  attributes{1}: LONGSET; (* MemReqs *)
                                  pri{2}: LONGINT;
                                  exec{8}: LONGINT;
                                  name{9}: ARRAY OF CHAR): LONGINT;
PROCEDURE AddPort* {exec,-354}(port{9}: MsgPortPtr);
PROCEDURE AddResource* {exec,-486}(resource{9}: ADDRESS);
PROCEDURE AddSemaphore* {exec,-600}(
                signalSemaphore{9}: SignalSemaphorePtr);
PROCEDURE AddTail* {exec,-246}(list{8}: ListPtr;
                               node{9}: ADDRESS);
PROCEDURE AddTask* {exec,-282}(task{9}: TaskPtr;
                               initialPC{10}: PROC;
```

```
                    finalPC{11}: ADDRESS);
PROCEDURE Alert* {exec,-108}(alertNum{7}: LONGINT;
                  parameters{13}: LONGINT);
PROCEDURE AllocAbs* {exec,-204}(byteSize{0}: LONGINT;
                    location{9}: LONGINT): ADDRESS;
PROCEDURE Allocate* {exec,-186}(freeList{8}: MemHeaderPtr;
                    byteSize{0}: LONGINT): ADDRESS;
PROCEDURE AllocEntry* {exec,-222}(
          memList1{8}: MemListPtr): ADDRESS;
PROCEDURE AllocMem* {exec,-198}(byteSize{0}: LONGINT;
                    requirements{1}: LONGSET (* MemReqs *)): ADDRESS;
PROCEDURE AllocSignal* {exec,-330}(
          signalNum1{0}: LONGINT): SHORTINT;
PROCEDURE AllocTrap* {exec,-342}(trapNum1{0}: LONGINT): SHORTINT;
PROCEDURE AttemptSemaphore* {exec,-576}(
          signalSemaphore{8}: SignalSemaphorePtr ): BOOLEAN;
PROCEDURE AvailMem* {exec,-216}(
          requirements{1}: LONGSET (* MemReqs *)): LONGINT;
PROCEDURE Cause* {exec,-180}(interrupt{9}: InterruptPtr);
PROCEDURE CheckIO* {exec,-468}(ioRequest{9}: ADDRESS): BOOLEAN;
PROCEDURE CloseDevice* {exec,-450}(ioRequest{9}: ADDRESS);
PROCEDURE CloseLibrary* {exec,-414}(library{9}: LibraryPtr);
PROCEDURE CopyMem* {exec,-624}(source{8}: ARRAY OF BYTE;
                   dest{9}: ARRAY OF BYTE;
                   size{0}: LONGINT);
PROCEDURE CopyMemQuick* {exec,-630}(source{8}: ARRAY OF BYTE;
                     dest{9}: ARRAY OF BYTE;
                     size{0}: LONGINT);
PROCEDURE Deallocate* {exec,-192}(freeList{8}: MemHeaderPtr;
                    memoryBlock{9}: ADDRESS;
                    byteSize{0}: LONGINT);
PROCEDURE Debug* {exec,-114}();
PROCEDURE Disable* {exec,-120}();
PROCEDURE Dispatch* {exec,- 60}();
PROCEDURE DoIO* {exec,-456}(ioRequest{9}: ADDRESS);
PROCEDURE Enable* {exec,-126}();
PROCEDURE Enqueue* {exec,-270}(list{8}: ListPtr;
                   node{9}: ADDRESS);
PROCEDURE Exception* {exec,- 66}();
PROCEDURE ExitIntr* {exec,- 36}();
PROCEDURE FindName* {exec,-276}(start{8}: ADDRESS;
                    name{9}: ARRAY OF CHAR): LONGINT;
PROCEDURE FindPort* {exec,-390}(name{9}: ARRAY OF CHAR): MsgPortPtr;
PROCEDURE FindResident* {exec,- 96}(name{9}: ARRAY OF CHAR): ResidentPtr;
PROCEDURE FindSemaphore* {exec,-594}(
          name{9}: ARRAY OF CHAR): SignalSemaphorePtr;
PROCEDURE FindTask* {exec,-294}(name{9}: ARRAY OF CHAR): TaskPtr;
PROCEDURE Forbid* {exec,-132}();
```

```
PROCEDURE FreeEntry* {exec,-228}(memList{8}: MemListPtr);
PROCEDURE FreeMem* {exec,-210}(memoryBlock{9}: ADDRESS;
                        byteSize{0}: LONGINT);
PROCEDURE FreeSignal* {exec,-336}(signalNum{0}: LONGINT);
PROCEDURE FreeTrap* {exec,-348}(trapNum{0}: LONGINT);
PROCEDURE GetCC* {exec,-528}(): SET;
PROCEDURE GetMsg* {exec,-372}(port{8}: MsgPortPtr): LONGINT;
PROCEDURE InitCode* {exec,- 72}(startClass{0}: SHORTSET; (* ResidentFlags *)
                         version{1}: LONGINT);
PROCEDURE InitResident* {exec,-102}(resident{9}: ResidentPtr;
                          segList{1}: BPTR);
PROCEDURE InitSemaphore* {exec,-558}(
                signalSemaphore{8}: SignalSemaphorePtr);
PROCEDURE InitStruct* {exec,- 78}(initTable{9}: ADDRESS;
                           memory{10}: ADDRESS;
                           size{0}: LONGINT);
PROCEDURE Insert* {exec,-234}(list{8}: ListPtr;
                       node{9}: ADDRESS;
                       listNode{10}: ADDRESS);
PROCEDURE MakeFunctions* {exec,- 90}(target{8}: ADDRESS;
                             functArray{9}: ADDRESS;
                             functDispBase{10}: ADDRESS);
PROCEDURE MakeLibrary* {exec,- 84}(vectors{8}: ADDRESS;
                            structure{9}: ADDRESS;
                            init{10}: ADDRESS;
                            dataSize{0}: LONGINT;
                            segList{1}: BPTR): LibraryPtr;
PROCEDURE ObtainSemaphore* {exec,-564}(
                signalSemaphore{8}: SignalSemaphorePtr);
PROCEDURE ObtainSemaphoreList* {exec,-582}(list{8}: ListPtr);
PROCEDURE OldOpenLibrary* {exec,-408}(
                libName{9}: ARRAY OF CHAR): LibraryPtr;
PROCEDURE OpenDevice* {exec,-444}(devName{8}: ARRAY OF CHAR;
                unitNumber{0}: LONGINT;
                ioRequest{9}: ADDRESS;
                flags{1}: LONGSET): LONGINT;
PROCEDURE OpenLibrary* {exec,-552}(libName{9}: ARRAY OF CHAR;
                            version{0}: LONGINT): LibraryPtr;
PROCEDURE OpenResource* {exec,-498}(
                    resName{9}: ARRAY OF CHAR): LONGINT;
PROCEDURE Permit* {exec,-138}();
PROCEDURE Procure* {exec,-540}(semaphore{8}: SemaphorePtr;
                        bidMessage{9}: MessagePtr): LONGINT;
PROCEDURE PutMsg* {exec,-366}(port{8}: MsgPortPtr;
                       message{9}: ADDRESS);
PROCEDURE RawDoFmt* {exec,-522}(formatString{8}: ARRAY OF CHAR;
                         dataStream{9}: ADDRESS;
                         putChProc{10}: PROC;
```

```
                                putChData{11}: ADDRESS);
PROCEDURE RawIOInit* {exec,-504}();
PROCEDURE RawMayGetChar* {exec,-510}(): CHAR;
PROCEDURE RawPutChar* {exec,-516}(ch{0}: CHAR);
PROCEDURE ReleaseSemaphore* {exec,-570}(
                  signalSemaphore{8}: SignalSemaphorePtr);
PROCEDURE ReleaseSemaphoreList* {exec,-588}(list{8}: ListPtr);
PROCEDURE RemDevice* {exec,-438}(device{9}: DevicePtr);
PROCEDURE RemHead* {exec,-258}(list{8}: ListPtr): ADDRESS;
PROCEDURE RemIntServer* {exec,-174}(intNum{0}: LONGINT;
                                  interrupt{9}: InterruptPtr);
PROCEDURE RemLibrary* {exec,-402}(library{9}: LibraryPtr);
PROCEDURE Remove* {exec,-252}(node{9}: ADDRESS);
PROCEDURE RemPort* {exec,-360}(port{9}: MsgPortPtr);
PROCEDURE RemResource* {exec,-492}(resource{9}: ADDRESS);
PROCEDURE RemSemaphore* {exec,-606}(
                  signalSemaphore{9}: SignalSemaphorePtr);
PROCEDURE RemTail* {exec,-264}(list{8}: ListPtr): ADDRESS;
PROCEDURE RemTask* {exec,-288}(task{9}: TaskPtr);
PROCEDURE ReplyMsg* {exec,-378}(message{9}: ADDRESS);
PROCEDURE Reschedule* {exec,- 48}();
PROCEDURE Schedule* {exec,- 42}();
PROCEDURE SendIO* {exec,-462}(ioRequest{9}: ADDRESS);
PROCEDURE SetExcept* {exec,-312}(newSignals{0}: LONGSET;
                            signalMask{1}: LONGSET): LONGSET;
PROCEDURE SetFunction* {exec,-420}(library{9}: LibraryPtr;
                              funcOffset{8}: INTEGER;
                              funcEntry{0}: ADDRESS): ADDRESS;
PROCEDURE SetIntVector* {exec,-162}(
                  intNumber{0}: LONGINT;
                  interrupt{9}: InterruptPtr): InterruptPtr;
PROCEDURE SetSignal* {exec,-306}(newSignals{0}: LONGSET;
                            signalMask{1}: LONGSET): LONGSET;
PROCEDURE SetSR* {exec,-144}(newSR{0}: SET;
                          mask{1}: SET): SET;
PROCEDURE SetTaskPri* {exec,-300}(task{9}: TaskPtr;
                              priority{0}: SHORTINT): SHORTINT;
PROCEDURE Signal* {exec,-324}(task{9}: TaskPtr;
                           signals{0}: LONGSET);
PROCEDURE SumKickData* {exec,-612}();
PROCEDURE SumLibrary* {exec,-426}(library{9}: LibraryPtr);
PROCEDURE SuperState* {exec,-150}(): LONGINT;
PROCEDURE Supervisor* {exec,- 30}();
PROCEDURE Switch* {exec,- 54}();
PROCEDURE TypeOfMem* {exec,-534}(
                     address{9}: ADDRESS): LONGSET (* MemReqs *);
PROCEDURE UserState* {exec,-156}(sysStack{0}: LONGINT);
PROCEDURE Vacate* {exec,-546}(semaphore{8}: SemaphorePtr);
```

```
PROCEDURE Wait* {exec,-318}(signalSet{0}: LONGSET): LONGSET;
PROCEDURE WaitIO* {exec,-474}(ioRequest{9}: ADDRESS);
PROCEDURE WaitPort* {exec,-384}(port{8}: MsgPortPtr);

END Exec.
```

ExecSupport:

```
DEFINITION ExecSupport;

TYPE
  PROC = PROCEDURE();

PROCEDURE NewList(VAR list: e.List);
PROCEDURE ListEmpty(VAR list: e.List): BOOLEAN;
PROCEDURE BeginIO(ioRequest: e.IORequestPtr);
PROCEDURE AbortIO(ioRequest: e.IORequestPtr);
PROCEDURE CreatePort(portName: LONGINT; priority: SHORTINT): e.MsgPortPtr;
PROCEDURE DeletePort(port: e.MsgPortPtr);
PROCEDURE CreateExtIO(ioReplyPort: e.MsgPortPtr; size: INTEGER): LONGINT;
PROCEDURE DeleteExtIO(extIOReq: LONGINT);
PROCEDURE CreateStdIO(ioReplyPort: e.MsgPortPtr): e.IOStdReqPtr;
PROCEDURE DeleteStdIO(ioStdReq: e.IOStdReqPtr);
PROCEDURE CreateTask(taskName: LONGINT;
                     priority: SHORTINT;
                     initPC: PROC;
                     stackSize: LONGINT): e.TaskPtr;
PROCEDURE DeleteTask(t: e.TaskPtr);

END ExecSupport.
```

Expansion:

```
MODULE Expansion;

IMPORT e: Exec,
       s: SYSTEM;

TYPE
  ExpansionRom * = STRUCT
    type * : BYTE;
    product * : BYTE;
    flags * : BYTE;
    reserved03 * : BYTE;
    manufacturer * : INTEGER;
    serialNumber * : LONGINT;
    initDiagVec * : INTEGER;
    reserved0c * : BYTE;
    reserved0d * : BYTE;
    reserved0e * : BYTE;
    reserved0f * : BYTE
  END;
  ExpansionControl * = STRUCT
    interrupt * : BYTE;
    reserved11 * : BYTE;
    baseAddress * : BYTE;
    shutup * : BYTE;
    reserved14 * : BYTE;
    reserved15 * : BYTE;
    reserved16 * : BYTE;
    reserved17 * : BYTE;
    reserved18 * : BYTE;
    reserved19 * : BYTE;
    reserved1a * : BYTE;
    reserved1b * : BYTE;
    reserved1c * : BYTE;
    reserved1d * : BYTE;
    reserved1e * : BYTE;
    reserved1f * : BYTE;
  END;

CONST
  slotSize * = 10000H;
  slotMask * = 0FFFFH;
  slotShift * = 16;
  expansionBase * = 0E80000H;
  expansionSize * = 080000H;
  expansionSlots * = 8;
```

```
  memoryBase * = 200000H;
  memorySize * = 800000H;
  memorySlots * = 128;
  typrMask * = 0C0H;
  typeBit * = 6;
  typeSize * = 2;
  newBoard * = 0C0H;
  memMask * = 07H;
  memBit * = 0;
  memSize * = 3;
  chainedConfig * = 3;
  diagValid * = 4;
  memList * = 5;
  memSpace * = 7;
  noShutup * = 6;

  intena * = 1;
  reset * = 3;
  int2pend * = 4;
  int6pend * = 5;
  int7pend * = 6;
  interrupting * = 7;

TYPE
  DiagArea * = STRUCT
    config * : BYTE;
    flags * : BYTE;
    size * : INTEGER;
    diagPoint * : INTEGER;
    bootPoint * : INTEGER;
    name * : INTEGER;
    reserved01 * : INTEGER;
    reserved02 * : INTEGER
  END;

CONST
  busWidth * = 0C0H;
  nibbleWide * = 0;
  byteWide * = 040H;
  wordWide * = 080H;
  bootTime * = 030H;
  never * = 0;
  configTime * = 010H;
  bindTime * = 020H;

TYPE
  ConfigDevPtr * = POINTER TO ConfigDev;
  ConfigDev * = STRUCT
```

```
    node * : e.Node;
    flags * : BYTE;
    pad * : BYTE;
    rom * : ExpansionRom;
    boardAddr * : e.ADDRESS;
    boardSize * : e.ADDRESS;
    slotAddr * : INTEGER;
    slotSize * : INTEGER;
    driver * : e.ADDRESS;
    nextCD * : ConfigDevPtr;
    unused * : ARRAY 4 OF LONGINT;
  END;

CONST
  shutup * = 0;
  configMe * = 1;

TYPE
  CurrentBinding * = STRUCT
    configDev * : ConfigDevPtr;
    fileName * : e.ADDRESS;
    productString * : e.ADDRESS;
    toolTypes * : e.ADDRESS;
  END;
  CurrentBindingPtr * = POINTER TO CurrentBinding;

VAR
  base * :  e.LibraryPtr;

PROCEDURE AddConfigDev * {base, -30}(configDev{8}:ConfigDevPtr);
PROCEDURE AddDosNode * {base, -150}(bootPri{0}:LONGINT;
                                    flags{1}:LONGSET;
                                    deviceNode{8}:e.ADDRESS):LONGINT;
PROCEDURE AllocBoardMem * {base, -42}(slotSpec{0}:LONGINT):LONGINT;
PROCEDURE AllocConfigDev * {base, -48}():ConfigDevPtr;
PROCEDURE AllocExpansionMem * {base, -54}(numSlots{0}:LONGINT;
                                          slotAlign{1}:LONGINT;
                                          slotOffset{2}:LONGINT):LONGINT;
PROCEDURE ConfigBoard * {base, -60}(board{8}:e.ADDRESS;
                                    configDev{9}:ConfigDevPtr):LONGINT;
PROCEDURE ConfigChain * {base, -66}(baseAddr{8}:e.ADDRESS);
PROCEDURE expansionUnused * {base, -36}();
PROCEDURE FindConfigDev * {base, -72}(oldConfigDev{8}:ConfigDevPtr;
                                      manufacturer{0}:LONGINT;
                                      product{1}:LONGINT);
PROCEDURE FreeBoardMem * {base, -78}(startSlot{0}:LONGINT;
                                     slotSpec{1}:LONGINT);
PROCEDURE FreeConfigDev * {base, -84}(configDev{8}:ConfigDevPtr);
```

```
PROCEDURE FreeExpansionMem * {base, -90}(startSlot{0}:LONGINT;
                                numSlots{1}:LONGINT);
PROCEDURE GetCurrentBinding * {base, -138}(
                    currentBinding{8}:CurrentBindingPtr;
                    size{0}:LONGINT):LONGINT;
PROCEDURE MakeDosNode * {base,-144}(
                    parameterPkt{0}:e.ADDRESS):e.ADDRESS;
PROCEDURE ObtainConfigBinding * {base, -120}();
PROCEDURE ReadExpansionByte * {base, -96}(board{8}:e.ADDRESS;
                                 offset{0}:LONGINT):BYTE;
PROCEDURE ReadExpansionRom * {base, -102}(
                    board{8}:e.ADDRESS;
                    configDev{9}:ConfigDevPtr):LONGINT;
PROCEDURE ReleaseConfigBinding * {base, -126}();
PROCEDURE RemConfigDev * {base, -108}(configDev{8}:ConfigDevPtr);
PROCEDURE SetCurrentBinding * {base, -132}(
                    currentBinding{8}:CurrentBindingPtr;
                                 size{0}:LONGINT);
PROCEDURE WriteExpansionByte * {base, -114}(board{8}:e.ADDRESS;
                                  offset{0}:LONGINT;
                                  byte{1}:BYTE);

(* $OvflChk- $RangeChk- $StackChk- $NilChk- $ReturnChk- $CaseChk- *)

BEGIN

  base :=  e.OpenLibrary(s.ADR("expansion.library"),33);
  IF base = NIL THEN HALT(0) END;

CLOSE

  IF base#NIL THEN e.CloseLibrary(base) END;

END Expansion.
```

GamePort:

```
MODULE GamePort; (* $Implementation- *)

IMPORT e: Exec;

CONST
  gamePortName * = "gameport.device";
  readEvent * = e.nonstd+0;
  askCType * = e.nonstd+1;
  setCType * = e.nonstd+2;
  askTrigger * = e.nonstd+3;
  setTrigger * = e.nonstd+4;
  allocated * = -1;
  errSetCType * = 1;

(* Keys *)
  downKeys * = 0;
  upKeys * = 1;

(* Controller *)
  noController * = 0;
  mouse * = 1;
  relJoystick * = 2;
  absJoystick * = 3;

TYPE
  GamePortTrigger * = STRUCT
    keys * : SET; (* Keys *)
    timeout * : INTEGER;
    xDelta * : INTEGER;
    yDelta * : INTEGER;
  END;

END GamePort.
```

Graphics:

```
MODULE Graphics;

IMPORT e: Exec,
       h: Hardware,
       s: SYSTEM;

TYPE
 AnimCompPtr * = POINTER TO AnimComp;
 AnimObPtr * = POINTER TO AnimOb;
 AreaInfoPtr * = POINTER TO AreaInfo;
 BitMapPtr * = POINTER TO BitMap;
 BobPtr * = POINTER TO Bob;
 ClipRectPtr * = POINTER TO ClipRect;
 CollTablePtr * = POINTER TO CollTable;
 ColorMapPtr * = POINTER TO ColorMap;
 CopinitPtr * = POINTER TO Copinit;
 CopInsPtr * = POINTER TO CopIns;
 CopListPtr * = POINTER TO CopList;
 CprlistPtr * = POINTER TO Cprlist;
 DBufPacketPtr * = POINTER TO DBufPacket;
 GelsInfoPtr * = POINTER TO GelsInfo;
 IsrvstrPtr * = POINTER TO Isrvstr;
 LayerPtr * = POINTER TO Layer;
 LayerInfoPtr * = POINTER TO LayerInfo;
 RasInfoPtr * = POINTER TO RasInfo;
 RastPortPtr * = POINTER TO RastPort;
 RectanglePtr * = POINTER TO Rectangle;
 RegionPtr * = POINTER TO Region;
 RegionRectanglePtr * = POINTER TO RegionRectangle;
 SimpleSpritePtr * = POINTER TO SimpleSprite;
 TextAttrPtr * = POINTER TO TextAttr;
 TextFontPtr * = POINTER TO TextFont;
 TmpRasPtr * = POINTER TO TmpRas;
 UCopListPtr * = POINTER TO UCopList;
 ViewPtr * = POINTER TO View;
 ViewPortPtr * = POINTER TO ViewPort;
 VSpritePtr * = POINTER TO VSprite;

 Rectangle * = STRUCT
  minX * : INTEGER;
  minY * : INTEGER;
  maxX * : INTEGER;
  maxY * : INTEGER;
 END;
 Layer * = STRUCT
```

```
  front * : LayerPtr;
  back * : LayerPtr;
  clipRect * : ClipRectPtr;
  rp * : RastPortPtr;
  bounds * : Rectangle;
  reserved * : ARRAY 4 OF BYTE;
  priority * : INTEGER;
  flags * : INTEGER;
  superBitMap * : BitMapPtr;
  superClipRect * : ClipRectPtr;
  window * : e.ADDRESS;
  scrollX * : INTEGER;
  scrollY * : INTEGER;
  cr * : ClipRectPtr;
  cr2 * : ClipRectPtr;
  crnew * : ClipRectPtr;
  superSaveClipRects * : ClipRectPtr;
  cliprects * : ClipRectPtr;
  layerInfo * : LayerInfoPtr;
  lock * : e.SignalSemaphore;
  reserved3 * : ARRAY 8 OF BYTE;
  clipRegion * : RegionPtr;
  saveClipRects * : RegionPtr;
  reserved2 * : ARRAY 22 OF BYTE;
  damageList * : RegionPtr;
 END;
 ClipRect * = STRUCT
  next * : ClipRectPtr;
  prev * : ClipRectPtr;
  lobs * : LayerPtr;
  bitMap * : BitMapPtr;
  bounds * : Rectangle;
  p1 * : ClipRectPtr;
  p2 * : ClipRectPtr;
  reserved * : LONGINT;
  flags * : LONGINT;
 END;

CONST
 needsNoConcealedRasters * = 01H;
 isLessX * = 1;
 isLessY * = 2;
 isGrtrX * = 4;
 isGrtrY * = 8;
 borderHit * = 0;
 topHit * = 1;
 bottomHit * = 2;
 leftHit * = 4;
```

```
 rightHit * = 8;

CONST
 move * = 0;
 wait * = 1;
 next * = 2;
 sht * = 14;
 lof * = 15;

TYPE
 CopIns * = STRUCT
  opCode * : INTEGER;
  destAddr * : INTEGER;  (* vWaitPos *)
  destData * : INTEGER;  (* hWaitPos *)
 END;
 CopInsCLPtr * = STRUCT  (* für opCode * = next *)
  opCode * : INTEGER;
  nxtlist * : CopListPtr;
 END;
 Cprlist * = STRUCT
  next * : CprlistPtr;
  start * : e.ADDRESS;
  maxCount * : INTEGER;
 END;
 CopList * = STRUCT
  next * : CopListPtr;
  copList * : CopListPtr;
  viewPort * : ViewPortPtr;
  copIns * : CopInsPtr;
  copPtr * : CopInsPtr;
  copLStart * : e.ADDRESS;
  copSStart * : e.ADDRESS;
  count * : INTEGER;
  maxCount * : INTEGER;
  dyOffset * : INTEGER;
 END;
 UCopList * = STRUCT
  next * : UCopListPtr;
  firstCopList * : CopListPtr;
  copList * : CopListPtr;
 END;
 Copinit * = STRUCT
  diagstrt * : ARRAY 4 OF INTEGER;
  sprstrtup * : ARRAY (2*8*2)+2+(2*2)+2 OF INTEGER;
  sprstop * : ARRAY 2 OF INTEGER;
 END;

CONST
```

```
interlace * = 04H;
pf2pri * = 40H;
colorOn * = 200H;
dblpf * = 400H;
holdnmodify * = 800H;
m640 * = 08000H;
plnCntMsk * = 07H;
plnCntShft * = 12;

fineScroll * = 0FH;
fineScrollShift * = 04H;
fineScrollMask * = 0FH;
vrtclPos * = 01FFH;
vrtclPosShift * = 07H;
horizPos * = 07FH;
dftchMask * = 0FFH;
vposrlof * = 08000H;

ringtrigger * = 01H;
anfracsize * = 06H;
animhalf * = 020H;

b2Norm * = 0;
b2Swap * = 1;
b2Bobber * = 2;

CONST
(* VSpriteFlags *)
  vsprite    * = 0;
  saveBack   * = 1;
  overlay    * = 2;
  mustDraw   * = 3;
  backSaved  * = 8;
  bobUpdate  * = 9;
  gelGone    * = 10;
  vsOverflow * = 11;

TYPE
 VSprite * = STRUCT
  nextVSprite * : VSpritePtr;
  prevVSprite * : VSpritePtr;
  drawPath * : VSpritePtr;
  clearPath * : VSpritePtr;
  oldY * : INTEGER;
  oldX * : INTEGER;
  flags * : SET; (* VSpriteFlags *)
  y * : INTEGER;
  x * : INTEGER;
```

```
  height * : INTEGER;
  width * : INTEGER;
  depth * : INTEGER;
  meMask * : SET;
  hitMask * : SET;
  imageData * : e.ADDRESS;
  borderLine * : e.ADDRESS;
  collMask * : e.ADDRESS;
  sprColors * : e.ADDRESS;
  vsBob * : BobPtr;
  planePick * : SHORTSET;
  planeOnOff * : SHORTSET;
 END;

CONST
(* BobFlag *)
  saveBob     * = 0;
  bobIsComp   * = 1;
  bWaiting    * = 8;
  bDrawn      * = 9;
  bobsAway    * = 10;
  bobNix      * = 11;
  savePreserve * = 12;
  outStep     * = 13;

TYPE
 Bob * = STRUCT
  flags * : SET; (* BobFlags *)
  saveBuffer * : e.ADDRESS;
  imageShadow * : e.ADDRESS;
  before * : BobPtr;
  after * : BobPtr;
  bobVSprite * : VSpritePtr;
  bobComp * : AnimCompPtr;
  dBuffer * : DBufPacketPtr;
 END;
 AnimComp * = STRUCT
  flags * : INTEGER;
  timer * : INTEGER;
  timeSet * : INTEGER;
  nextComp * : AnimCompPtr;
  prevComp * : AnimCompPtr;
  nextSeq * : AnimCompPtr;
  prevSeq * : AnimCompPtr;
  animCRoutine * : e.ADDRESS;
  yTrans * : INTEGER;
  xTrans * : INTEGER;
  headOb * : AnimObPtr;
```

```
 animBob * : BobPtr;
END;
AnimOb * = STRUCT
 nextOb * : AnimObPtr;
 prevOb * : AnimObPtr;
 clock * : LONGINT;
 anOldY * : INTEGER;
 anOldX * : INTEGER;
 anY * : INTEGER;
 anX * : INTEGER;
 yVel * : INTEGER;
 xVel * : INTEGER;
 yAccel * : INTEGER;
 xAccel * : INTEGER;
 ringYTrans * : INTEGER;
 ringXTrans * : INTEGER;
 animORoutine * : e.ADDRESS;
 headComp * : AnimCompPtr;
END;
DBufPacket * = STRUCT
 bufY * : INTEGER;
 bufX * : INTEGER;
 bufPath * : VSpritePtr;
 bufBuffer * : e.ADDRESS;
END;
CollTable * = STRUCT
 collPtrs * : ARRAY 16 OF e.ADDRESS
END;
BitMap * = STRUCT
 bytesPerRow * : INTEGER;
 rows * : INTEGER;
 flags * : BYTE;
 depth * : BYTE;
 pad * : INTEGER;
 planes * : ARRAY 8 OF e.ADDRESS;
END;

CONST
(* DisplayFlags *)
 ntsc   * = 0;
 genloc * = 1;
 pal    * = 2;

TYPE
 GfxBase * = STRUCT
  libNode * : e.Library;
  actiView * : ViewPtr;
  copinit * : CopinitPtr;
```

```
  cia * : e.ADDRESS;
  blitter * : e.ADDRESS;
  loFlist * : e.ADDRESS;
  shFlist * : e.ADDRESS;
  blthd * : h.BltnodePtr;
  blttl * : h.BltnodePtr;
  bsblthd * : h.BltnodePtr;
  bsblttl * : h.BltnodePtr;
  vbsrv * : e.Interrupt;
  timsrv * : e.Interrupt;
  bltsrv * : e.Interrupt;
  textFonts * : e.List;
  defaultFont * : TextFontPtr;
  modes * : SET;
  vBlank * : BYTE;
  debug * : BYTE;
  beamSync * : INTEGER;
  bplcon0 * : SET;
  spriteReserved * : BYTE;
  bytereserved * : BYTE;
  flags * : SET;
  blitLock * : INTEGER;
  blitNest * : INTEGER;
  blitWaitQ * : e.List;
  blitOwner * : e.TaskPtr;
  waitQ * : e.List;
  displayFlags * : SHORTSET; (* DisplayFlags *)
  simpleSprites * : e.ADDRESS;
  maxDisplayRow * : INTEGER;
  maxDisplayColumn * : INTEGER;
  normalDisplayRows * : INTEGER;
  normalDisplayColumns * : INTEGER;
  normalDPMX * : INTEGER;
  normalDPMY * : INTEGER;
  lastChanceMemory * : e.SignalSemaphorePtr;
  lcMptr * : e.ADDRESS;
  microsPerLine * : INTEGER;
  minDisplayColumn * : INTEGER;
  reserved * : ARRAY 23 OF LONGINT;
 END;
 GfxBasePtr * = POINTER TO GfxBase;

CONST
 blitMsgFault * = 4;

TYPE
 Isrvstr * = STRUCT
  node * : e.Node;
```

```
  iptr * : IsrvstrPtr;
  code * : e.ADDRESS;
  ccode * : e.ADDRESS;
  carg * : LONGINT;
 END;

CONST
(* LayerFlags *)
 layerSimple       * = 0;
 layerSmart        * = 1;
 layerSuper        * = 2;
 layerUpdating     * = 4;
 layerBackdrop     * = 6;
 layerRefresh      * = 7;
 layerClipRectsLost * = 8;

TYPE
 LayerInfo * = STRUCT
  layer * : LayerPtr;
  lp * : LayerPtr;
  obs * : LayerPtr;
  freeClipRects * : e.MinList;
  lock * : e.SignalSemaphore;
  head * : e.List;
  longreserved * : LONGINT;
  flags * : SET; (* LayerFlags *)
  count * : SHORTINT;
  lockLayersCount * : SHORTINT;
  layerInfoExtraSize * : INTEGER;
  blitbuff * : e.ADDRESS;
  layerInfoExtra * : e.ADDRESS;
 END;

CONST
 lmnRegion * = -1;
 newLayerInfoCalled * = 01H;
 alertLayersNoMem * = MIN(LONGINT) + 3010000H; (* 083010000H *)

TYPE
 AreaInfo * = STRUCT
  vctrTbl * : e.ADDRESS;
  vctrPtr * : e.ADDRESS;
  flagTbl * : e.ADDRESS;
  flagPtr * : e.ADDRESS;
  count * : INTEGER;
  maxCount * : INTEGER;
  firstX * : INTEGER;
  firstY * : INTEGER;
```

```
 END;
 TmpRas * = STRUCT
  rasPtr * : e.ADDRESS;
  size * : LONGINT;
 END;
 GelsInfo * = STRUCT
  sprRsrvd * : SHORTINT;
  flags * : BYTE;
  gelHead * : VSpritePtr;
  gelTail * : VSpritePtr;
  nextLine * : e.ADDRESS;
  lastColor * : e.ADDRESS;
  collHandler * : CollTablePtr;
  leftmost * : INTEGER;
  rightmost * : INTEGER;
  topmost * : INTEGER;
  bottommost * : INTEGER;
  firstBlissObj * : e.ADDRESS;
  lastBlissObj * : e.ADDRESS;
 END;

CONST
(* DrawModes *)
 complement * = 1;
 inversvid  * = 2;
 jam1 * = SHORTSET{};
 jam2 * = SHORTSET{0};

(* FontStyles *)
 underlined  * = 0;
 bold        * = 1;
 italic      * = 2;
 extended    * = 3;
 normalFont * = SHORTSET{};

(* FontFlags *)
 romFont     * = 0;
 diskFont    * = 1;
 revPath     * = 2;
 tallDot     * = 3;
 wideDot     * = 4;
 proportional * = 5;
 designed    * = 6;
 removed     * = 7;

(* RastPortFlags *)
 firstDot    * = 0;
 oneDot      * = 1;
```

```
 dBuffer     * = 2;
 areaOutline * = 3;
 noCrossFill * = 5;

 spriteAttached * = 080H;

TYPE
 Point * = STRUCT
  x * ,y * : INTEGER;
 END;
 RastPort * = STRUCT
  layer * : LayerPtr;
  bitMap * : BitMapPtr;
  areaPtrn * : e.ADDRESS;
  tmpRas * : TmpRasPtr;
  areaInfo * : AreaInfoPtr;
  gelsInfo * : GelsInfoPtr;
  mask * : SHORTSET;
  fgPen * : BYTE;
  bgPen * : BYTE;
  aOlPen * : BYTE;
  drawMode * : SHORTSET; (* DrawModes *)
  areaPtSz * : BYTE;
  linPatCnt * : BYTE;
  dummy * : BYTE;
  flags * : SET; (* RastPortFlags *)
  linePtrn * : INTEGER;
  x * : INTEGER;
  y * : INTEGER;
  minterms * : ARRAY 8 OF BYTE;
  penWidth * : INTEGER;
  penHeight * : INTEGER;
  font * : TextFontPtr;
  algoStyle * : SHORTSET; (* FontStyles *)
  txFlags * : SHORTSET; (* FontFlags *)
  txHeight * : INTEGER;
  txWidth * : INTEGER;
  txBaseline * : INTEGER;
  txSpacing * : INTEGER;
  user * : e.ADDRESS;
  longreserved * : ARRAY 2 OF LONGINT;
  wordreserved * : ARRAY 7 OF INTEGER;
  reserved * : ARRAY 8 OF BYTE;
 END;

TYPE
 RegionRectangle * = STRUCT
  next * : RegionRectanglePtr;
```

```
 prev * : RegionRectanglePtr;
 bounds * : Rectangle;
END;
Region * = STRUCT
 bounds * : Rectangle;
 regionRectangle * : RegionRectanglePtr;
END;
SimpleSprite * = STRUCT
 posctldata * : e.ADDRESS;
 height * : INTEGER;
 x * : INTEGER;
 y * : INTEGER;
 num * : INTEGER;
END;
TextAttr * = STRUCT
 name * : e.ADDRESS;
 ySize * : INTEGER;
 style * : SHORTSET; (* FontStyleSet *)
 flags * : SHORTSET; (* FontFlags *)
END;
TextFont * = STRUCT
 message * : e.Message;
 ySize * : INTEGER;
 style * : SHORTSET; (* FontStyles *)
 flags * : SHORTSET; (* FontFlags *)
 xSize * : INTEGER;
 baseline * : INTEGER;
 boldSmear * : INTEGER;
 accessors * : INTEGER;
 loChar * : CHAR;
 hiChar * : CHAR;
 charData * : e.ADDRESS;
 modulo * : INTEGER;
 charLoc * : e.ADDRESS;
 charSpace * : e.ADDRESS;
 charKern * : e.ADDRESS;
END;
ColorMap * = STRUCT
 flags * : BYTE;
 type * : BYTE ;
 count * : INTEGER;
 colorTable * : e.ADDRESS;
END;

CONST
(* ViewModes *)
 genlocVideo   * = 1;
 lace          * = 2;
```

```
 pfba          * = 6;
 extraHalfbrite * = 7;
 genlocAudio    * = 8;
 dualpf        * = 10;
 ham           * = 11;
 vpHide         * = 13;
 sprites       * = 14;
 hires         * = 15;

TYPE
 ViewPort * = STRUCT
  next * : ViewPortPtr;
  colorMap * : ColorMapPtr;
  dspIns * : CopListPtr;
  sprIns * : CopListPtr;
  clrIns * : CopListPtr;
  uCopIns * : UCopListPtr;
  dWidth * : INTEGER;
  dHeight * : INTEGER;
  dxOffset * : INTEGER;
  dyOffset * : INTEGER;
  modes * : SET; (* ViewModes *)
  spritePriorities * : BYTE;
  reserved * : BYTE;
  rasInfo * : RasInfoPtr;
 END;
 View * = STRUCT
  viewPort * : ViewPortPtr;
  lofCprList * : CprlistPtr;
  shfCprList * : CprlistPtr;
  dyOffset * : INTEGER;
  dxOffset * : INTEGER;
  modes * : SET; (* ViewModes *)
 END;
 RasInfo * = STRUCT
  next * : RasInfoPtr;
  bitMap * : BitMapPtr;
  rxOffset * : INTEGER;
  ryOffset * : INTEGER;
 END;

 PROC * = PROCEDURE();

VAR
  gfx * :  GfxBasePtr;

(*-----------------------------------------------------------------------*)
```

```
(* $OvflChk- $RangeChk- $StackChk- $NilChk- $ReturnChk- $CaseChk- *)

(*------ graphics.library: ------*)

PROCEDURE AddAnimOb*{gfx,-156}(   anOb{8}:AnimObPtr;
                                  VAR anKey{9}:AnimObPtr;
                                     rp{10}:RastPortPtr);
PROCEDURE AddBob*{gfx,- 96}(Bob{8}:BobPtr;
                              rp{9}:RastPortPtr);
PROCEDURE AddFont*{gfx,-480}(textFont{9}:TextFontPtr);
PROCEDURE AddVSprite*{gfx,-102}(vs{8}:VSpritePtr;
                                 rp{9}:RastPortPtr);
PROCEDURE AllocRaster*{gfx,-492}(width{0}:INTEGER;
                                  height{1}:INTEGER):e.ADDRESS;
PROCEDURE AndRectRegion*{gfx,-504}(region{8}:RegionPtr;
                                    rectang{9}:RectanglePtr);
PROCEDURE AndRegionRegion*{gfx,-624}(region1{8}:RegionPtr;
                                      region2{9}:RegionPtr):BOOLEAN;
PROCEDURE Animate*{gfx,-162}(VAR anKey{8}:AnimObPtr; rp{9}:RastPortPtr);
PROCEDURE AreaDraw*{gfx,-258}(rp{9}:RastPortPtr;
                               x{0}:INTEGER;
                               y{1}:INTEGER):BOOLEAN;
PROCEDURE AreaEllipse*{gfx,-186}(rp{9}:RastPortPtr;
                                  cX{0}:INTEGER;
                                  cY{1}:INTEGER;
                                  a{2}:INTEGER;
                                  b{3}:INTEGER):BOOLEAN;
PROCEDURE AreaEnd*{gfx,-264}(rp{9}:RastPortPtr):BOOLEAN;
PROCEDURE AreaMove*{gfx,-252}(rp{9}:RastPortPtr;
                               x{0}:INTEGER;
                               y{1}:INTEGER):BOOLEAN;
PROCEDURE AskFont*{gfx,-474}(rp{9}:RastPortPtr;
                              textAttr{8}:TextAttrPtr);
PROCEDURE AskSoftStyle*{gfx,- 84}(
                        rp{9}:RastPortPtr):SHORTSET; (* FontStyleSet *)
PROCEDURE AttemptLockLayerRom*{gfx,-654}(layer{13}:LayerPtr):BOOLEAN;
PROCEDURE BltBitMap*{gfx,- 30}(srcBitMap{8}:BitMapPtr;
                                srcX{0}:INTEGER;
                                srcY{1}:INTEGER;
                                dstBitMap{9}:BitMapPtr;
                                dstX{2}:INTEGER;
                                dstY{3}:INTEGER;
                                sizeX{4}:INTEGER;
                                sizeY{5}:INTEGER;
                                minterm{6}:BYTE;
```

```
                    mask{7}:SHORTSET;
                    tempA{10}:e.ADDRESS):LONGINT;
PROCEDURE BltBitMapRastPort*{gfx,-606}(srcbm{8}:BitMapPtr;
                    srcX{0}:INTEGER;
                    srcY{1}:INTEGER;
                    destRp{9}:RastPortPtr;
                    destX{2}:INTEGER;
                    destY{3}:INTEGER;
                    sizeX{4}:INTEGER;
                    sizeY{5}:INTEGER;
                    minterm{6}:BYTE);
PROCEDURE BltClear*{gfx,-300}(memBlock{9}:e.ADDRESS;
                    bytecount{0}:LONGINT;
                    flags{1}:LONGSET);
PROCEDURE BltMaskBitMapRastPort*{gfx,-636}(srcbm{8}:BitMapPtr;
                    srcX{0}:INTEGER;
                    srcY{1}:INTEGER;
                    destRp{9}:RastPortPtr;
                    destX{2}:INTEGER;
                    destY{3}:INTEGER;
                    sizeX{4}:INTEGER;
                    sizeY{5}:INTEGER;
                    minterm{6}:BYTE;
                    bltmask{10}:e.ADDRESS);
PROCEDURE BltPattern*{gfx,-312}(rp{9}:RastPortPtr;
                    mask{8}:e.ADDRESS;
                    xl{0}:INTEGER;
                    yl{1}:INTEGER;
                    maxX{2}:INTEGER;
                    maxY{3}:INTEGER;
                    bytecnt{4}:INTEGER);
PROCEDURE BltTemplate*{gfx,- 36}(srcTemplate{8}:e.ADDRESS;
                    srcX{0}:INTEGER;
                    srcMod{1}:INTEGER;
                    rp{9}:RastPortPtr;
                    dstX{2}:INTEGER;
                    dstY{3}:INTEGER;
                    sizeX{4}:INTEGER;
                    sizeY{5}:INTEGER);
PROCEDURE CBump*{gfx,-366}(c{9}:UCopListPtr);
PROCEDURE ChangeSprite*{gfx,-420}(vp{8}:ViewPortPtr;
                    s{9}:SimpleSpritePtr;
                    newdata{10}:e.ADDRESS);
PROCEDURE ClearEOL*{gfx,- 42}(rp{9}:RastPortPtr);
PROCEDURE ClearRectRegion*{gfx,-522}(region{8}:RegionPtr;
                    rectangle{9}:RectanglePtr):BOOLEAN;
PROCEDURE ClearRegion*{gfx,-528}(region{8}:RegionPtr);
PROCEDURE ClearScreen*{gfx,- 48}(rp{9}:RastPortPtr);
```

```
PROCEDURE ClipBlit*{gfx,-552}(src{8}:RastPortPtr;
                              srcX{0}:INTEGER;
                              srcY{1}:INTEGER;
                              dest{9}:RastPortPtr;
                              destX{2}:INTEGER;
                              destY{3}:INTEGER;
                              xSize{4}:INTEGER;
                              ySize{5}:INTEGER;
                              minterm{6}:BYTE);
PROCEDURE CloseFont*{gfx,- 78}(font{9}:TextFontPtr);
PROCEDURE CMove*{gfx,-372}(c{9}:UCopListPtr;
                           a{0}:e.ADDRESS;
                           v{1}:INTEGER);
PROCEDURE CopySBitMap*{gfx,-450}(layer{8}:LayerPtr);
PROCEDURE CWait*{gfx,-378}(c{9}:UCopListPtr;
                           v{0}:INTEGER;
                           h{1}:INTEGER);
PROCEDURE DisownBlitter*{gfx,-462}();
PROCEDURE DisposeRegion*{gfx,-534}(region{8}:RegionPtr);
PROCEDURE DoCollision*{gfx,-108}(rp{9}:RastPortPtr);
PROCEDURE Draw*{gfx,-246}(rp{9}:RastPortPtr;
                          x{0}:INTEGER;
                          y{1}:INTEGER);
PROCEDURE DrawEllipse*{gfx,-180}(rp{9}:RastPortPtr;
                                 cX{0}:INTEGER;
                                 cY{1}:INTEGER;
                                 a{2}:INTEGER;
                                 b{3}:INTEGER);
PROCEDURE DrawGList*{gfx,-114}(rp{9}:RastPortPtr;
                               vp{8}:ViewPortPtr);
PROCEDURE Flood*{gfx,-330}(rp{9}:RastPortPtr;
                           mode{2}:LONGINT;
                           x{0}:INTEGER;
                           y{1}:INTEGER):BOOLEAN;
PROCEDURE FreeColorMap*{gfx,-576}(colorMap{8}:ColorMapPtr);
PROCEDURE FreeCopList*{gfx,-546}(coplist{8}:CopListPtr);
PROCEDURE FreeCprList*{gfx,-564}(cprlist{8}:CprlistPtr);
PROCEDURE FreeGBuffers*{gfx,-600}(anOb{8}:AnimObPtr;
                                  rp{9}:RastPortPtr;
                                  db{0}:BOOLEAN);
PROCEDURE FreeRaster*{gfx,-498}(p{8}:e.ADDRESS;
                                width{0}:INTEGER;
                                height{1}:INTEGER);
PROCEDURE FreeSprite*{gfx,-414}(pick{0}:INTEGER);
PROCEDURE FreeVPortCopLists*{gfx,-540}(vp{8}:ViewPortPtr);
PROCEDURE GetColorMap*{gfx,-570}(entries{0}:LONGINT):ColorMapPtr;
PROCEDURE GetGBuffers*{gfx,-168}(anOb{8}:AnimObPtr;
                                 rp{9}:RastPortPtr;
```

```
                    db{0}:BOOLEAN):BOOLEAN;
PROCEDURE GetRGB4*{gfx,-582}(colorMap{8}:ColorMapPtr;
                  entry{0}:LONGINT):LONGINT;
PROCEDURE GetSprite*{gfx,-408}(sprite{8}:SimpleSpritePtr;
                   pick{0}:INTEGER):INTEGER;
PROCEDURE GraphicsReserved1*{gfx,-642}():LONGINT;
PROCEDURE GraphicsReserved2*{gfx,-648}():LONGINT;
PROCEDURE InitArea*{gfx,-282}(VAR areainfo{8}:AreaInfo;
                     buffer{9}:e.ADDRESS;
                     maxvectors{0}:INTEGER);
PROCEDURE InitBitMap*{gfx,-390}(VAR bm{8}:BitMap;
                       depth{0}:INTEGER;
                       width{1}:INTEGER;
                       height{2}:INTEGER);
PROCEDURE InitGels*{gfx,-120}(head{8}:VSpritePtr;
                  tail{9}:VSpritePtr;
                  gInfo{10}:GelsInfoPtr);
PROCEDURE InitGMasks*{gfx,-174}(anOb{8}:AnimObPtr);
PROCEDURE InitMasks*{gfx,-126}(vs{8}:VSpritePtr);
PROCEDURE InitRastPort*{gfx,-198}(VAR rp{9}:RastPort);
PROCEDURE InitTmpRas*{gfx,-468}(VAR tmpras{8}:TmpRas;
                       buffer{9}:e.ADDRESS;
                       size{0}:LONGINT);
PROCEDURE InitView*{gfx,-360}(VAR view{9}:View);
PROCEDURE InitVPort*{gfx,-204}(VAR vp{8}:ViewPort);
PROCEDURE LoadRGB4*{gfx,-192}(vp{8}:ViewPortPtr;
                  colors{9}:ARRAY OF INTEGER;
                  count{0}:INTEGER);
PROCEDURE LoadView*{gfx,-222}(view{9}:ViewPtr);
PROCEDURE LockLayerRom*{gfx,-432}(layer{13}:LayerPtr);
PROCEDURE MakeVPort*{gfx,-216}(view{8}:ViewPtr;
                   viewport{9}:ViewPortPtr);
PROCEDURE Move*{gfx,-240}(rp{9}:RastPortPtr;
                x{0}:INTEGER;
                y{1}:INTEGER);
PROCEDURE MoveSprite*{gfx,-426}(vp{8}:ViewPortPtr;
                  sprite{9}:SimpleSpritePtr;
                  x{0}:INTEGER;
                  y{1}:INTEGER);
PROCEDURE MrgCop*{gfx,-210}(view{9}:ViewPtr);
PROCEDURE NewRegion*{gfx,-516}():RegionPtr;
PROCEDURE OpenFont*{gfx,- 72}(textAttr{8}:TextAttrPtr):TextFontPtr;
PROCEDURE OrRectRegion*{gfx,-510}(region{8}:RegionPtr;
                    rectangle{9}:RectanglePtr):BOOLEAN;
PROCEDURE OrRegionRegion*{gfx,-612}(region1{8}:RegionPtr;
                     region2{9}:RegionPtr):BOOLEAN;
PROCEDURE OwnBlitter*{gfx,-456}();
PROCEDURE PolyDraw*{gfx,-336}(rp{9}:RastPortPtr;
```

```
                        count{0}:INTEGER;
                        array{8}:ARRAY OF Point);
PROCEDURE QBlit*{gfx,-276}(bp{9}:h.BltnodePtr);
PROCEDURE QBSBlit*{gfx,-294}(bsp{9}:h.BltnodePtr);
PROCEDURE ReadPixel*{gfx,-318}(rp{9}:RastPortPtr;
                         x{0}:INTEGER;
                         y{1}:INTEGER):LONGINT;
PROCEDURE RectFill*{gfx,-306}(rp{9}:RastPortPtr;
                        xMin{0}:INTEGER;
                        yMin{1}:INTEGER;
                        xMax{2}:INTEGER;
                        yMax{3}:INTEGER);
PROCEDURE RemFont*{gfx,-486}(textFont{9}:TextFontPtr);
PROCEDURE RemIBob*{gfx,-132}(bob{8}:BobPtr;
                       rp{9}:RastPortPtr;
                       vp{10}:ViewPortPtr);
PROCEDURE RemVSprite*{gfx,-138}(vs{8}:VSpritePtr);
PROCEDURE ScrollRaster*{gfx,-396}(rp{9}:RastPortPtr;
                            dx{0}:INTEGER;
                            dy{1}:INTEGER;
                            xMin{2}:INTEGER;
                            yMin{3}:INTEGER;
                            xMax{4}:INTEGER;
                            yMax{5}:INTEGER);
PROCEDURE ScrollVPort*{gfx,-588}(vp{8}:ViewPortPtr);
PROCEDURE SetAPen*{gfx,-342}(rp{9}:RastPortPtr;
                       pen{0}:INTEGER);
PROCEDURE SetBPen*{gfx,-348}(rp{9}:RastPortPtr;
                       pen{0}:INTEGER);
PROCEDURE SetCollision*{gfx,-144}(num{0}:LONGINT;
                            routine{8}:PROC;
                            gInfo{9}:GelsInfoPtr);
PROCEDURE SetDrMd*{gfx,-354}(rp{9}:RastPortPtr;
                      mode{0}:SHORTSET (* DrawModes *));
PROCEDURE SetFont*{gfx,- 66}(rp{9}:RastPortPtr;
                      font{8}:TextFontPtr);
PROCEDURE SetRast*{gfx,-234}(rp{9}:RastPortPtr;
                      pen{0}:INTEGER);
PROCEDURE SetRGB4*{gfx,-288}(vp{8}:ViewPortPtr;
                      n{0}:INTEGER;
                      r{1}:INTEGER;
                      g{2}:INTEGER;
                      b{3}:INTEGER);
PROCEDURE SetRGB4CM*{gfx,-630}(cm{8}:ColorMapPtr;
                        n{0}:INTEGER;
                        r{1}:INTEGER;
                        g{2}:INTEGER;
                        b{3}:INTEGER);
```

```
PROCEDURE SetSoftStyle*{gfx,- 90}(
        rp{9}:RastPortPtr;
        style{0}:SHORTSET; (* FontStyles *)
        enable{1}:SHORTSET (* FontStyles *)):SHORTSET; (* FontStyles *)
PROCEDURE SortGList*{gfx,-150}(rp{9}:RastPortPtr);
PROCEDURE SyncSBitMap*{gfx,-444}(layer{8}:LayerPtr);
PROCEDURE Text*{gfx,- 60}(rp{9}:RastPortPtr;
                    string{8}:ARRAY OF CHAR;
                    count{0}:LONGINT);
PROCEDURE TextLength*{gfx,- 54}(rp{9}:RastPortPtr;
                        string{8}:ARRAY OF CHAR;
                        count{0}:LONGINT):INTEGER;
PROCEDURE UCopperListInit*{gfx,-594}(copperList{8}:UCopListPtr;
                            num{0}:LONGINT):UCopListPtr;
PROCEDURE UnlockLayerRom*{gfx,-438}(layer{13}:LayerPtr);
PROCEDURE VBeamPos*{gfx,-384}():LONGINT;
PROCEDURE WaitBlit*{gfx,-228}();
PROCEDURE WaitBOVP*{gfx,-402}(vp{8}:ViewPortPtr);
PROCEDURE WaitTOF*{gfx,-270}();
PROCEDURE WritePixel*{gfx,-324}(rp{9}:RastPortPtr;
                x{0}:INTEGER;
                y{1}:INTEGER):BOOLEAN;
PROCEDURE XorRectRegion*{gfx,-558}(region{8}:RegionPtr;
                        rectangle{9}:RectanglePtr):BOOLEAN;
PROCEDURE XorRegionRegion*{gfx,-618}(region1{8}:RegionPtr;
                            region2{9}:RegionPtr):BOOLEAN;


(*-----------------------------------------------------------------------*)


(*------ GfxMacros: ------*)


PROCEDURE OnDisplay*;
BEGIN h.custom.dmacon := h.bitSet + {h.raster} END OnDisplay;

PROCEDURE OffDisplay*;
BEGIN h.custom.dmacon := h.bitClr + {h.raster} END OffDisplay;

PROCEDURE OnSprite*;
BEGIN h.custom.dmacon := h.bitSet + {h.sprite} END OnSprite;

PROCEDURE OffSprite*;
BEGIN h.custom.dmacon := h.bitClr + {h.sprite} END OffSprite;

PROCEDURE OnVBlank*;
BEGIN h.custom.intena := h.bitSet + {e.vertb} END OnVBlank;
```

```
PROCEDURE OffVBlank*;
BEGIN h.custom.intena := h.bitClr + {e.vertb} END OffVBlank;

PROCEDURE SetOPen*(w: RastPortPtr; c: BYTE);
BEGIN
  w.aOlPen := c;
  INCL(w.flags,areaOutline);
END SetOPen;

PROCEDURE SetDrPt*(w: RastPortPtr; p: INTEGER);
BEGIN
  w.linePtrn := p;
  INCL(w.flags,firstDot);
END SetDrPt;

PROCEDURE SetWrMsk*(w: RastPortPtr; m: SHORTSET);
BEGIN w.mask := m END SetWrMsk;

PROCEDURE SetAfPt*(w: RastPortPtr; p: e.ADDRESS; n: BYTE);
BEGIN
  w.areaPtrn := p;
  w.areaPtSz := n;
END SetAfPt;

PROCEDURE BndryOff*(w: RastPortPtr);
BEGIN EXCL(w.flags,areaOutline) END BndryOff;

PROCEDURE CINIT*(c: UCopListPtr; n: LONGINT);
BEGIN IF UCopperListInit(c,n)=NIL THEN END END CINIT;

PROCEDURE CMOVE*(c: UCopListPtr; a: e.ADDRESS; b: INTEGER);
BEGIN CMove(c,a,b); CBump(c) END CMOVE;

PROCEDURE CWAIT*(c: UCopListPtr; a,b: INTEGER);
BEGIN CWait(c,a,b); CBump(c) END CWAIT;

PROCEDURE CEND*(c: UCopListPtr);
BEGIN CWAIT(c,10000,255) END CEND;


(*-----------------------------------------------------------------------*)


BEGIN

 gfx := s.VAL(GfxBasePtr,e.OpenLibrary("graphics.library",33));
 IF gfx = NIL THEN HALT(0) END;
```

```
CLOSE
 IF gfx#NIL THEN e.CloseLibrary(s.VAL(e.LibraryPtr,gfx)) END;
END Graphics.
```

Hardware:

```
MODULE Hardware; (* $Implementation- *)

IMPORT e: Exec;

CONST
(* AdkFlags *)
  use0v1   * = 0;
  use1v2   * = 1;
  use2v3   * = 2;
  use3vn   * = 3;
  use0p1   * = 4;
  use1p2   * = 5;
  use2p3   * = 6;
  use3pn   * = 7;
  fast     * = 8;
  msbSync  * = 9;
  wordSync * = 10;
  uartBrk  * = 11;
  mfmPrec  * = 12;
  preComp0 * = 13;
  preComp1 * = 14;
  adkSet   * = 15;

  pre000ns * = {};
  pre140ns * = {preComp0};
  pre280ns * = {preComp1};
  pre560ns * = {preComp0,preComp1};

TYPE
  AudioInfo * = STRUCT
    acptr * : e.ADDRESS;
    aclen * : INTEGER;
    acper * : INTEGER;
    acvol * : INTEGER;
    acdat * : INTEGER;
    acpad * : ARRAY 2 OF INTEGER;
  END;
  AudioChannels * = ARRAY 4 OF AudioInfo;

CONST
(* BC0Flags *)
  nanbnc * = 0;
  nanbc  * = 1;
  nabnc  * = 2;
  nabc   * = 3;
```

```
  anbnc  * = 4;
  anbc   * = 5;
  abnc   * = 6;
  abc    * = 7;
  dest   * = 8;
  srcC   * = 9;
  srcB   * = 10;
  srcA   * = 11;
  ash1   * = 12;
  ash2   * = 13;
  ash4   * = 14;
  ash8   * = 15;

  aORb * = {abc,anbc,nabc,abnc,anbnc,nabnc};
  aORc * = {abc,nabc,abnc,anbc,nanbc,anbnc};
  aXORc * = {nabc,abnc,nanbc,anbnc};
  aTOd * = {abc,anbc,abnc,anbnc};

(* BC1Flags *)
  lineMode    * = 0;
  desc        * = 1;
  fillCarryIn * = 2;
  fillOr      * = 3;
  fillXor     * = 4;
  ovFlag      * = 5;
  signFlag    * = 6;
  bsh1        * = 12;
  bsh2        * = 13;
  bsh4        * = 14;
  bsh8        * = 15;

  oneDot * = desc;
  blitReverse * = desc;
  aul * = {fillCarryIn};
  sul * = {fillOr};
  sud * = {fillXor};
  octant1 * = sud;
  octant2 * = {};
  octant3 * = sul;
  octant4 * = aul+sud;
  octant5 * = aul+sul+sud;
  octant6 * = aul+sul;
  octant7 * = aul;
  octant8 * = sul+sud;

(* BPLC0Flags *)
  ersy  * = 1;
  lace  * = 2;
```

```
  lpen  * = 3;
  gaud  * = 8;
  color * = 9;
  dblpf * = 10;
  homod * = 11;
  bpu0  * = 12;
  bpu1  * = 13;
  bpu2  * = 14;
  hires * = 15;

TYPE
  BltnodePtr * = POINTER TO Bltnode;
  Bltnode * = STRUCT
    n * : BltnodePtr;
    function * : e.ADDRESS;
    stat * : CHAR;
    blitsize * : INTEGER;
    beamsync * : INTEGER;
    cleanup * : e.ADDRESS;
  END;

CONST
  cleanup * = 40H;
  cleanme * = cleanup;

(* CollisionControlFlags *)
  plane1       * = 0;
  plane2       * = 1;
  plane3       * = 2;
  plane4       * = 3;
  plane5       * = 4;
  plane6       * = 5;
  enablePlane1   * = 6;
  enablePlane2   * = 7;
  enablePlane3   * = 8;
  enablePlane4   * = 9;
  enablePlane5   * = 10;
  enablePlane6   * = 11;
  enableSprite01 * = 12;
  enableSprite23 * = 13;
  enableSprite45 * = 14;
  enableSprite67 * = 15;

(* CollisionFlags *)
  play1toPlay2      * = 0;
  play1toSprite01   * = 1;
  play1toSprite23   * = 2;
  play1toSprite45   * = 3;
```

```
  play1toSprite67   * = 4;
  play2toSprite01   * = 5;
  play2toSprite23   * = 6;
  play2toSprite45   * = 7;
  play2toSprite67   * = 8;
  sprite01toSprite23 * = 9;
  sprite01toSprite45 * = 10;
  sprite01toSprite67 * = 11;
  sprite23toSprite45 * = 12;
  sprite23toSprite67 * = 13;
  sprite45toSprite67 * = 14;

(* DiskFlags *)
  df0      * = 0;
  df1      * = 1;
  df2      * = 2;
  df3      * = 3;
  wordEqual * = 4;
  diskWrite * = 5;
  dmaOn     * = 6;
  dskByte   * = 7;

TYPE
  Coord * = STRUCT
    v * ,h * : SHORTINT;
  END;
  DiskInfo * = STRUCT
    flags * : SHORTSET; (* DiskFlags *)
    data * : BYTE;
  END;

CONST
(* DmaFlags *)
  aud0     * = 0;
  aud1     * = 1;
  aud2     * = 2;
  aud3     * = 3;
  disk     * = 4;
  sprite   * = 5;
  blitter  * = 6;
  copper   * = 7;
  raster   * = 8;
  master   * = 9;
  blithog  * = 10;
  bltnzero * = 13;
  bltdone  * = 14;
  dmaSet   * = 15;
```

```
  dmaAll * = {aud0..raster};
  bitSet * = {dmaSet};
  bitClr * = {};

CONST
  hSizeBits * = 6;
  vSizeBits * = 16-hSizeBits;
  hSizeMask * = 03FH;
  vSizeMask * = 03FFH;
  maxBytesPerRow * = 128;

(* PotFlags *)
  start  * = 0;
  datalx  * = 8;
  outlx  * = 9;
  dataly  * = 10;
  outly  * = 11;
  datarx  * = 12;
  outrx  * = 13;
  datary  * = 14;
  outry  * = 15;

(* SerialFlags *)
  d8   * = 0;
  stop  * = 1;
  sf2  * = 2;
  rxd  * = 3;
  tsre  * = 4;
  tbe  * = 5;
  rbf  * = 6;
  ovrun * = 7;

TYPE
  SerialInfo * = STRUCT
    flags * : SHORTSET; (* SerialFlags *)
    data * : CHAR;
  END;

CONST
(* SpriteControlFlags *)
  sho  * = 0;
  ev8  * = 1;
  sv8  * = 2;
  sct3 * = 3;
  sct4 * = 4;
  sct5 * = 5;
  sct6 * = 6;
  att  * = 7;
```

```
TYPE
  SpriteControlInfo * = STRUCT
    ev * : BYTE;
    flags * : SHORTSET; (* SpriteControlFlags *)
  END;
  SpriteInfo * = STRUCT
    pos * : INTEGER;
    ctl * : SpriteControlInfo;
    data * : LONGINT;
  END;
  Sprites * = ARRAY 8 OF SpriteInfo;

  Custom * = STRUCT
    bltddat * : INTEGER;
    dmaconr * : SET; (* DmaFlags *)
    vposr * : LONGINT;
    dskdatr * : INTEGER;
    joy0dat * : Coord;
    joy1dat * : Coord;
    clxdat * : SET; (* CollisionFlags *)
    adkconr * : SET; (* AdkFlags *)
    pot0dat * : Coord;
    pot1dat * : Coord;
    potinp * : SET; (* PotFlags *)
    serdatr * : SerialInfo;
    dskbytr * : DiskInfo;
    intenar * : SET; (* Exec.IntFlags *)
    intreqr * : SET; (* Exec.IntFlags *)
    dskpt * : e.ADDRESS;
    dsklen * : INTEGER;
    dskdat * : INTEGER;
    refptr * : INTEGER;
    vposw * : LONGINT;
    copcon * : SET;
    serdat * : SerialInfo;
    serper * : INTEGER;
    potgo * : SET; (* PotFlags *)
    joytest * : Coord;
    strequ * : INTEGER;
    strvbl * : INTEGER;
    strhor * : INTEGER;
    strlong * : INTEGER;
    bltcon0 * : SET; (* BC0Flags *)
    bltcon1 * : SET; (* BC1Flags *)
    bltafwm * : SET;
    bltalwm * : SET;
    bltcpt * : e.ADDRESS;
```

```
    bltbpt * : e.ADDRESS;
    bltapt * : e.ADDRESS;
    bltdpt * : e.ADDRESS;
    bltsize * : INTEGER;
    pad2d * : ARRAY 3 OF INTEGER;
    bltcmod * : INTEGER;
    bltbmod * : INTEGER;
    bltamod * : INTEGER;
    bltdmod * : INTEGER;
    pad34 * : ARRAY 4 OF INTEGER;
    bltcdat * : INTEGER;
    bltbdat * : INTEGER;
    bltadat * : INTEGER;
    pad3b * : ARRAY 4 OF INTEGER;
    dsksync * : INTEGER;
    cop1lc * : e.ADDRESS;
    cop2lc * : e.ADDRESS;
    copjmp1 * : INTEGER;
    copjmp2 * : INTEGER;
    copins * : INTEGER;
    diwstrt * : Coord;
    diwstop * : Coord;
    ddfstrt * : Coord;
    ddfstop * : Coord;
    dmacon * : SET; (* DmaFlags *)
    clxcon * : SET; (* CollisionControlFlags *);
    intena * : SET; (* Exec.IntFlags *)
    intreq * : SET; (* Exec.IntFlags *)
    adkcon * : SET; (* AdkFlags *)
    aud * : AudioChannels;
    bplpt * : ARRAY 6 OF e.ADDRESS;
    pad7c * : ARRAY 4 OF INTEGER;
    bplcon0 * : SET; (* BPLC0Flags *)
    bplcon1 * : INTEGER;
    bplcon2 * : INTEGER;
    pad83 * : INTEGER;
    bpl1mod * : INTEGER;
    bpl2mod * : INTEGER;
    pad86 * : ARRAY 2 OF INTEGER;
    bpldat * : ARRAY 6 OF INTEGER;
    pad8e * : ARRAY 2 OF INTEGER;
    sprpt * : ARRAY 8 OF e.ADDRESS;
    spr * : Sprites;
    color * : ARRAY 32 OF INTEGER;
  END;

VAR
  custom * [0DFF000H] : Custom;
```

```
CONST
(* CiaIcrFlags *)
  ta     * = 0;
  tb     * = 1;
  alrm   * = 2;
  sp     * = 3;
  flg    * = 4;
  setclr * = 7;

  ir * = setclr; (* On read setclr has the meaning of ir *)

(* CiaCraFlags *)
  craStart   * = 0;
  craPbon    * = 1;
  craOutmode * = 2;
  craRunmode * = 3;
  craLoad    * = 4;
  craInmode  * = 5;
  craSpmode  * = 6;
  craTodin   * = 7;

(* CiaCrbFlags *)
  crbStart   * = 0;
  crbPbon    * = 1;
  crbOutmode * = 2;
  crbRunmode * = 3;
  crbLoad    * = 4;
  crbInmode0 * = 5;
  crbInmode1 * = 6;
  crbAlarm   * = 7;

(* CiaaPraFlags *)
  overlay    * = 0;
  led        * = 1;
  dskChange  * = 2;
  dskProt    * = 3;
  dskTrack0  * = 4;
  dskRdy     * = 5;
  gamePort0  * = 6;
  gamePort1  * = 7;

(* CiabPraFlags *)
  prtrBusy * = 0;
  prtrPOut * = 1;
  prtrSel  * = 2;
  comDSR   * = 3;
  comCTS   * = 4;
```

```
  comCD   * = 5;
  comRTS  * = 6;
  comDTR  * = 7;

(* CiabPrbFlags *)
  dskStep  * = 0;
  dskDirec * = 1;
  dskSide  * = 2;
  dskSel0  * = 3;
  dskSel1  * = 4;
  dskSel2  * = 5;
  dskSel3  * = 6;
  dskMotor * = 7;

TYPE
  Pad * = ARRAY 254 OF SHORTSET;
  CIAA * = STRUCT
    pra * : SHORTSET; (* CiaaPraFlags *) pad0 * : Pad;
    prb * : SHORTSET;                   pad1 * : Pad;
    ddra * : SHORTSET; (* CiaaPraFlags *) pad2 * : Pad;
    ddrb * : SHORTSET; (* CiaaPrbFlags *) pad3 * : Pad;
    talo * : BYTE; pad4 * : Pad;
    tahi * : BYTE; pad5 * : Pad;
    tblo * : BYTE; pad6 * : Pad;
    tbhi * : BYTE; pad7 * : Pad;
    todlow * : BYTE; pad8 * : Pad;
    todmid * : BYTE; pad9 * : Pad;
    todhi * : BYTE; pad10 * : Pad;
    unusedreg * : SHORTSET; pad11 * : Pad;
    sdr * : SHORTSET; pad12 * : Pad;
    icr * : SHORTSET; (* CiaIcrFlags *) pad13 * : Pad;
    cra * : SHORTSET; (* CiaCraFlags *) pad14 * : Pad;
    crb * : SHORTSET; (* CiaCrbFlags *)
  END;
  CIAB * = STRUCT
    pra * : SHORTSET; (* CiabPraFlags *) pad0 * : Pad;
    prb * : SHORTSET; (* CiabPrbFlags *) pad1 * : Pad;
    ddra * : SHORTSET; (* CiabPraFlags *) pad2 * : Pad;
    ddrb * : SHORTSET; (* CiabPrbFlags *) pad3 * : Pad;
    talo * : BYTE; pad4 * : Pad;
    tahi * : BYTE; pad5 * : Pad;
    tblo * : BYTE; pad6 * : Pad;
    tbhi * : BYTE; pad7 * : Pad;
    todlow * : BYTE; pad8 * : Pad;
    todmid * : BYTE; pad9 * : Pad;
    todhi * : BYTE; pad10 * : Pad;
    unusedreg * : SHORTSET; pad11 * : Pad;
    sdr * : SHORTSET; pad12 * : Pad;
```

```
    icr * : SHORTSET; (* CiaIcrFlags *) pad13 * : Pad;
    cra * : SHORTSET; (* CiaCraFlags *) pad14 * : Pad;
    crb * : SHORTSET; (* CiaCrbFlags *)
  END;

VAR
  ciaa * [0BFE001H] : CIAA;
  ciab * [0BFD000H] : CIAB;

END Hardware.
```

Icon:

```
MODULE Icon;

IMPORT e: Exec,
       wb: Workbench,
       I: Intuition,
       s: SYSTEM;

VAR
  base * : e.LibraryPtr;

PROCEDURE AddFreeList * {base, -72}(
                    free{8}:wb.FreeListPtr; mem{9}:e.ADDRESS;
                    len{10}:LONGINT):BOOLEAN;
PROCEDURE AllocWBObject * {base, -66}():e.ADDRESS;
PROCEDURE BumpRevision * {base, -108}(new{8},old{9}: ARRAY OF CHAR);
PROCEDURE FreeDiskObject * {base, -90}(obj{8}:wb.DiskObjectPtr);
PROCEDURE FreeFreeList * {base, -54}(free{8}:wb.FreeListPtr);
PROCEDURE FreeWBObject * {base, -60}(obj{8}:e.ADDRESS);
PROCEDURE FindToolType * {base, -96}(
          toolTypes{8}: e.ADDRESS;
          typeName{9}: ARRAY OF CHAR):e.ADDRESS;
PROCEDURE GetDiskObject * {base, -78}(
          name{8}: ARRAY OF CHAR):wb.DiskObjectPtr;
PROCEDURE GetIcon * {base, -42}(name{8}: ARRAY OF CHAR;
                          icon{9}:wb.DiskObjectPtr;
                          f{10}:wb.FreeListPtr):LONGINT;
PROCEDURE GetWBObject * {base, -30}(
          name{8}: ARRAY OF CHAR):e.ADDRESS;
PROCEDURE MatchToolValue * {base, -102}(
          typeString{8},val{9}: ARRAY OF CHAR):BOOLEAN;
PROCEDURE PutDiskObject * {base, -84}(
          name{8}: ARRAY OF CHAR;
          obj{9}:wb.DiskObjectPtr):BOOLEAN;
PROCEDURE PutIcon * {base, -48}(
          name{8}: ARRAY OF CHAR;
          obj{9}:wb.DiskObjectPtr):LONGINT;
PROCEDURE PutWBObject * {base, -36}(
          name{8}: ARRAY OF CHAR;
          obj{9}:e.ADDRESS):LONGINT;

(* $OvflChk- $RangeChk- $StackChk- $NilChk- $ReturnChk- $CaseChk- *)

BEGIN

 base := e.OpenLibrary("icon.library",33);
```

```
 IF base=NIL THEN
   s.SETREG(0,I.DisplayAlert(0,"x00x64x14missing icon.library!",50));
   HALT(0)
 END;

CLOSE

 IF base#NIL THEN e.CloseLibrary(base) END;

END Icon.
```

Input:

```
MODULE Input; (* $Implementation- *)

IMPORT e * : Exec;

CONST
  inputName * = "input.device";
  addHandler * = e.nonstd+0;
  remHandler * = e.nonstd+1;
  writeEvent * = e.nonstd+2;
  setThresh * = e.nonstd+3;
  setPeriod * = e.nonstd+4;
  setMPort * = e.nonstd+5;
  setMType * = e.nonstd+6;
  setMTrig * = e.nonstd+7;

END Input.
```

InputEvent:

```
MODULE InputEvent; (* $Implementation- *)

IMPORT Exec, Timer;

CONST
(* Class *)
  null          * = 0;
  rawkey        * = 1;
  rawmouse      * = 2;
  event         * = 3;
  pointerpos    * = 4;
  timer         * = 6;
  gadgetdown    * = 7;
  gadgetup      * = 8;
  requester     * = 9;
  menulist      * = 10;
  closewindow   * = 11;
  sizewindow    * = 12;
  refreshwindow * = 13;
  newprefs      * = 14;
  diskremoved   * = 15;
  diskinserted  * = 16;
  activewindow  * = 17;
  inactivewindow * = 18;

  classMax      * = 18;
  upPrefix * = 080H;
  keyCodeFirst * = 00H;
  keyCodeLast * = 077H;
  commCodeFirst * = 078H;
  commCodeLast * = 07FH;
  c0First * = 00H;
  c0Last * = 01FH;
  asciiFirst * = 020H;
  asciiLast * = 07EH;
  asciiDel * = 07FH;
  c1First * = 080H;
  c1Last * = 09FH;
  latin1First * = 0A0H;
  latin1Last * = 0FFH;
  lButton * = 068H;
  rButton * = 069H;
  mButton * = 06AH;
  noButton * = 0FFH;
  newActive * = 01H;
```

```
  reqClear * = 00H;
  reqSet * = 01H;

(* Qualifiers *)
  lShift        * = 0;
  rShift        * = 1;
  capsLock      * = 2;
  control       * = 3;
  lAlt          * = 4;
  rAlt          * = 5;
  lCommand      * = 6;
  rCommand      * = 7;
  numericPad    * = 8;
  repeat        * = 9;
  interrupt     * = 10;
  multiBroadcast * = 11;
  midButton     * = 12;
  rightButton   * = 13;
  leftButton    * = 14;
  relativeMouse * = 15;

TYPE
  InputEventPtr * = POINTER TO InputEvent;
  InputEvent * = STRUCT
    nextEvent * : InputEventPtr;
    class * : SHORTINT; (* Class *)
    subClass * : SHORTINT; (* Class *)
    code * : INTEGER;
    qualifier * : SET; (* Qualifiers *)
    x * ,y * : INTEGER;
    timeStamp * : Timer.TimeVal
  END;
  InputEventAdr * = STRUCT
    nextEvent * : InputEventPtr;
    class * : SHORTINT; (* Class *)
    subClass * : SHORTINT; (* Class *)
    code * : INTEGER;
    qualifier * : SET; (* Qualifiers *)
    eventAddress * : Exec.ADDRESS;
    timeStamp * : Timer.TimeVal
  END;

END InputEvent.
```

Intuition:

```
MODULE Intuition;

IMPORT e:  Exec,
       g:  Graphics,
       ie: InputEvent,
           Timer,
           KeyMap,
       s:  SYSTEM;

TYPE
  BorderPtr * = POINTER TO Border;
  GadgetPtr * = POINTER TO Gadget;
  ImagePtr * = POINTER TO Image;
  IntuiMessagePtr * = POINTER TO IntuiMessage;
  IntuiTextPtr * = POINTER TO IntuiText;
  MenuItemPtr * = POINTER TO MenuItem;
  MenuPtr * = POINTER TO Menu;
  PreferencesPtr * = POINTER TO Preferences;
  PropInfoPtr * = POINTER TO PropInfo;
  RememberPtr * = POINTER TO Remember;
  RequesterPtr * = POINTER TO Requester;
  ScreenPtr * = POINTER TO Screen;
  StringInfoPtr * = POINTER TO StringInfo;
  WindowPtr * = POINTER TO Window;

CONST
  menuEnabled * = 0;
  miDrawn * = 8;

TYPE
  Menu * = STRUCT
    nextMenu * : MenuPtr;
    leftEdge * : INTEGER;
    topEdge * : INTEGER;
    width * : INTEGER;
    height * : INTEGER;
    flags * : SET;
    menuName * : e.ADDRESS;
    firstItem * : MenuItemPtr;
    jazzX * : INTEGER;
    jazzY * : INTEGER;
    beatX * : INTEGER;
    beatY * : INTEGER;
  END;
```

```
CONST
(* MenuItemFlags *)
  checkIt     * = 0;
  itemText    * = 1;
  commSeq     * = 2;
  menuToggle  * = 3;
  itemEnabled * = 4;
  highComp    * = 6;
  highBox     * = 7;
  checked     * = 8;
  isDrawn     * = 12;
  highItem    * = 13;
  menuToggled * = 14;

  highNone * = {highBox,highComp};
  checkWidth * = 19;
  commWidth * = 27;
  lowCheckWidth * = 13;
  lowCommWidth * = 16;

TYPE
  MenuItem * = STRUCT
    nextItem * : MenuItemPtr;
    leftEdge * : INTEGER;
    topEdge * : INTEGER;
    width * : INTEGER;
    height * : INTEGER;
    flags * : SET; (* MenuuItemFlags *)
    mutualExclude * : LONGSET;
    itemFill * : e.ADDRESS;
    selectFill * : e.ADDRESS;
    command * : CHAR;
    subItem * : MenuItemPtr;
    nextSelect * : INTEGER;
  END;

CONST
(* RequesterFlags *)
  pointRel     * = 0;
  preDrawn     * = 1;
  noisyReq     * = 2;
  reqOffWindow * = 12;
  reqActive    * = 13;
  sysRequest   * = 14;
  deferRefresh * = 15;

TYPE
  Requester * = STRUCT
```

```
    olderRequest * : RequesterPtr;
    leftEdge * : INTEGER;
    topEdge * : INTEGER;
    width * : INTEGER;
    height * : INTEGER;
    relLeft * : INTEGER;
    relTop * : INTEGER;
    reqGadget * : GadgetPtr;
    reqBorder * : BorderPtr;
    reqText * : IntuiTextPtr;
    flags * : SET; (* RequesterFlags *)
    backFill * : BYTE;
    reqLayer * : g.LayerPtr;
    reqPad1 * : ARRAY 32 OF SHORTINT;
    imageBMap * : g.BitMapPtr;
    rWindow * : WindowPtr;
    reqPad2 * : ARRAY 36 OF SHORTINT;
  END;

CONST
(* GadgetFlags *)
  gadgHBox     * = 0;
  gadgHImage   * = 1;
  gadgImage    * = 2;
  gRelBottom   * = 3;
  gRelRight    * = 4;
  gRelWidth    * = 5;
  gRelHeight   * = 6;
  selected     * = 7;
  gadgDisabled * = 8;

(* ActivationFlags *)
  relVerify    * = 0;
  gadgImmediate * =  1;
  endGadget    * = 2;
  followMouse  * = 3;
  rightBorder  * = 4;
  leftBorder   * = 5;
  topBorder    * = 6;
  bottomBorder * = 7;
  toggleSelect * = 8;
  stringCenter * = 9;
  stringRight  * = 10;
  longint      * = 11;
  altKeyMap    * = 12;
  boolExtend   * = 13;

  gadgHighbits * = {gadgHBox..gadgHImage};
```

```
  gadgHNone * = {gadgHBox,gadgHImage};
  gadgHComp * = {};
  boolGadget * = 0001H;
  gadget0002 * = 0002H;
  propGadget * = 0003H;
  strGadget * = 0004H;
  sizing * = 0010H;
  wDragging * = 0020H;
  sDragging * = 0030H;
  wUpFront * = 0040H;
  sUpFront * = 0050H;
  wDownBack * = 0060H;
  sDownBack * = 0070H;
  close * = 0080H;
  reqGadget * = 1000H;
  gzzGadget * = 2000H;
  scrGadget * = 4000H;
  sysGadget * =-8000H; (* sorry für das Vorzeichen, es gibt ja keine CARDINALs... *)
  gadgetType * = {10..15};

TYPE
  Gadget * = STRUCT
    nextGadget * : GadgetPtr;
    leftEdge * : INTEGER;
    topEdge * : INTEGER;
    width * : INTEGER;
    height * : INTEGER;
    flags * : SET; (* GadgetFlags *)
    activation * : SET; (* ActivationFlags *)
    gadgetType * : INTEGER;
    gadgetRender * : e.ADDRESS;
    selectRender * : e.ADDRESS;
    gadgetText * : IntuiTextPtr;
    mutualExclude * : LONGSET;
    specialInfo * : e.ADDRESS;
    gadgetID * : INTEGER;
    userData * : e.ADDRESS;
  END;

CONST
  boolMask * = 1;

TYPE
  BoolInfo * = STRUCT
    flags * : SET;
    mask * : e.ADDRESS;
    reserved * : LONGINT;
  END;
```

```
CONST
(* PropInfoFlags *)
  autoKnob      * = 0;
  freeHoriz     * = 1;
  freeVert      * = 2;
  propBorderless * = 3;
  knobHit       * = 8;

  knobVmin * = 4;
  knobHmin * = 6;
  maxBody * = -1;
  maxPot * = -1;

TYPE
  PropInfo * = STRUCT
    flags * : SET; (* PropInfoFlags *)
    horizPot * : INTEGER;
    vertPot * : INTEGER;
    horizBody * : INTEGER;
    vertBody * : INTEGER;
    cWidth * : INTEGER;
    cHeight * : INTEGER;
    hPotRes * : INTEGER;
    vPotRes * : INTEGER;
    leftBorder * : INTEGER;
    topBorder * : INTEGER;
  END;

  StringInfo * = STRUCT
    buffer * : e.ADDRESS;
    undoBuffer * : e.ADDRESS;
    bufferPos * : INTEGER;
    maxChars * : INTEGER;
    dispPos * : INTEGER;
    undoPos * : INTEGER;
    numChars * : INTEGER;
    dispCount * : INTEGER;
    cLeft * : INTEGER;
    cTop * : INTEGER;
    layerPtr * : g.LayerPtr;
    longInt * : LONGINT;
    altKeyMap * : KeyMap.KeyMapPtr;
  END;

CONST
  autoFrontPen * = 0;
  autoBackPen * = 1;
```

```
  autoDrawMode * = g.jam2;
  autoLeftEdge * = 6;
  autoTopEdge * = 3;
  autoITextFont * = NIL;
  autoNextText * = NIL;

TYPE
  IntuiText * = STRUCT
    frontPen * : SHORTINT;
    backPen * : SHORTINT;
    drawMode * : SHORTSET; (* g.DrawModes *)
    leftEdge * : INTEGER;
    topEdge * : INTEGER;
    iTextFont * : g.TextAttrPtr;
    iText * : e.ADDRESS;
    nextText * : IntuiTextPtr;
  END;

  Border * = STRUCT
    leftEdge * : INTEGER;
    topEdge * : INTEGER;
    frontPen * : SHORTINT;
    backPen * : SHORTINT;
    drawMode * : SHORTSET; (* g.DrawModes *)
    count * : SHORTINT;
    xy * : e.ADDRESS;
    nextBorder * : BorderPtr;
  END;

  Image * = STRUCT
    leftEdge * : INTEGER;
    topEdge * : INTEGER;
    width * : INTEGER;
    height * : INTEGER;
    depth * : INTEGER;
    imageData * : e.ADDRESS;
    planePick * : SHORTSET;
    planeOnOff * : SHORTSET;
    nextImage * : ImagePtr;
  END;

CONST
(* IDCMPFlags *)
  sizeVerify   * = 0;
  newSize      * = 1;
  refreshWindow * = 2;
  mouseButtons  * = 3;
  mouseMove    * = 4;
```

```
gadgetDown    * = 5;
gadgetUp      * = 6;
reqSet        * = 7;
menuPick      * = 8;
closeWindow   * = 9;
rawKey        * = 10;
reqVerify     * = 11;
reqClear      * = 12;
menuVerify    * = 13;
newPrefs      * = 14;
diskInserted  * = 15;
diskRemoved   * = 16;
wbenchMessage * = 17;
activeWindow  * = 18;
inactiveWindow * = 19;
deltaMove     * = 20;
vanillaKey    * = 21;
intuiTicks    * = 22;
lonelyMessage * = 31;

selectUp * = ie.lButton+ie.upPrefix;   (* mouseButtons *)
selectDown * = ie.lButton;
menuUp * = ie.rButton+ie.upPrefix;
menuDown * = ie.rButton;
menuNull * = -1;              (* menuPick *)
noMenu * = 1FH; noItem * = 3FH; noSub * = 1FH;
keyCodeQ * = 10H;             (* rawKey *)
keyCodeX * = 32H;
keyCodeV * = 34H;
keyCodeB * = 35H;
keyCodeN * = 36H;
keyCodeM * = 37H;
cursorUp * = 4CH;
cursorDown * = 4DH;
cursorRight * = 4EH;
cursorLeft * = 4FH;
menuHot * = 1;                (* menuVerify *)
menuCancel * = 2;
menuWaiting * = 3;
okOk * = menuHot;
okAbort * = 4;
okCancel * = menuCancel;
wbenchOpen * = 1;             (* wbenchMessage *)
wbenchClose * = 2;
(* IntuiMessage.qualifier *)
altLeft * = {ie.lAlt};
altRight * = {ie.rAlt};
amigaLeft * = {ie.lCommand};
```

```
  amigaRight * = {ie.rCommand};
  amigaKeys * = amigaLeft+amigaRight;

TYPE
  IntuiMessage * = STRUCT
    execMessage * : e.Message;
    class * : LONGSET; (* IDCMPFlags *)
    code * : INTEGER;
    qualifier * : SET; (* ie.Qualifiers *)
    iAddress * : e.ADDRESS;
    mouseX * : INTEGER;
    mouseY * : INTEGER;
    time * : Timer.TimeVal;
    idcmpWindow * : WindowPtr;
    specialLink * : IntuiMessagePtr;
  END;

CONST
(* WindowFlags *)
  windowSizing  * = 0;
  windowDrag    * = 1;
  windowDepth   * = 2;
  windowClose   * = 3;
  sizeBRight    * = 4;
  sizeBBottom   * = 5;
  simpleRefresh * = 6;
  superBitMap   * = 7;
  backDrop      * = 8;
  reportMouse   * = 9;
  gimmeZeroZero * = 10;
  borderless    * = 11;
  activate      * = 12;
  windowActive  * = 13;
  inRequest     * = 14;
  menuState     * = 15;
  rmbTrap       * = 16;
  noCareRefresh * = 17;
  windowRefresh * = 24;
  wbenchWindow  * = 25;
  windowTicked  * = 26;

  otherRefresh * = LONGSET{simpleRefresh,superBitMap};
  superUnused * = LONGSET{18..23,27..31};

(* ScreenFlags *)
  wbenchScreen * = 0;
  showTitle    * = 4;
```

```
  beeping      * = 5;
  customBitMap * = 6;
  screenBehind * = 7;
  screenQuiet  * = 8;

  stdScreenHeight * = -1;
  customScreen * = {wbenchScreen..3};

TYPE
  NewWindow * = STRUCT
    leftEdge * : INTEGER;
    topEdge * : INTEGER;
    width * : INTEGER;
    height * : INTEGER;
    detailPen * : SHORTINT;
    blockPen * : SHORTINT;
    idcmpFlags * : LONGSET; (* IDCMPFlags *)
    flags * : LONGSET; (* WindowFlags *)
    firstGadget * : GadgetPtr;
    checkMark * : ImagePtr;
    title * : e.ADDRESS;
    screen * : ScreenPtr;
    bitMap * : g.BitMapPtr;
    minWidth * : INTEGER;
    minHeight * : INTEGER;
    maxWidth * : INTEGER;
    maxHeight * : INTEGER;
    type * : SET; (* ScreenFlags *)
  END;
  NewScreen * = STRUCT
    leftEdge * : INTEGER;
    topEdge * : INTEGER;
    width * : INTEGER;
    height * : INTEGER;
    depth * : INTEGER;
    detailPen * : SHORTINT;
    blockPen * : SHORTINT;
    viewModes * : SET; (* g.ViewModes *)
    type * : SET; (* ScreenFlags *)
    font * : g.TextAttrPtr;
    defaultTitle * : e.ADDRESS;
    gadgets * : GadgetPtr;
    customBitMap * : g.BitMapPtr;
  END;
  Window * = STRUCT
    nextWindow * : WindowPtr;
    leftEdge * : INTEGER;
    topEdge * : INTEGER;
```

```
  width * : INTEGER;
  height * : INTEGER;
  mouseY * : INTEGER;
  mouseX * : INTEGER;
  minWidth * : INTEGER;
  minHeight * : INTEGER;
  maxWidth * : INTEGER;
  maxHeight * : INTEGER;
  flags * : LONGSET; (* WindowFlags *)
  menuStrip * : MenuPtr;
  title * : e.ADDRESS;
  firstRequest * : RequesterPtr;
  dmRequest * : RequesterPtr;
  reqCount * : INTEGER;
  wScreen * : ScreenPtr;
  rPort * : g.RastPortPtr;
  borderLeft * : SHORTINT;
  borderTop * : SHORTINT;
  borderRight * : SHORTINT;
  borderBottom * : SHORTINT;
  borderRPort * : g.RastPortPtr;
  firstGadget * : GadgetPtr;
  parent * : WindowPtr;
  descendant * : WindowPtr;
  pointer * : e.ADDRESS;
  ptrHeight * : SHORTINT;
  ptrWidth * : SHORTINT;
  xOffset * : SHORTINT;
  yOffset * : SHORTINT;
  idcmpFlags * : LONGSET; (* IDCMPFlags *)
  userPort * : e.MsgPortPtr;
  windowPort * : e.MsgPortPtr;
  messageKey * : IntuiMessagePtr;
  detailPen * : SHORTINT;
  blockPen * : SHORTINT;
  checkMark * : ImagePtr;
  screenTitle * : e.ADDRESS;
  gzzMouseX * : INTEGER;
  gzzMouseY * : INTEGER;
  gzzWidth * : INTEGER;
  gzzHeight * : INTEGER;
  extData * : e.ADDRESS;
  userData * : e.ADDRESS;
  wLayer * : g.LayerPtr;
  iFont * : g.TextFontPtr;
END;
Screen * = STRUCT
  nextScreen * : ScreenPtr;
```

```
    firstWindow * : WindowPtr;
    leftEdge * : INTEGER;
    topEdge * : INTEGER;
    width * : INTEGER;
    height * : INTEGER;
    mouseY * : INTEGER;
    mouseX * : INTEGER;
    flags * : SET; (* ScreenFlags *)
    title * : e.ADDRESS;
    defaultTitle * : e.ADDRESS;
    barHeight * : SHORTINT;
    barVBorder * : SHORTINT;
    barHBorder * : SHORTINT;
    menuVBorder * : SHORTINT;
    menuHBorder * : SHORTINT;
    wBorTop * : SHORTINT;
    wBorLeft * : SHORTINT;
    wBorRight * : SHORTINT;
    wBorBottom * : SHORTINT;
    font * : g.TextAttrPtr;
    viewPort * : g.ViewPort;
    rastPort * : g.RastPort;
    bitMap * : g.BitMap;
    layerInfo * : g.LayerInfo;
    firstGadget * : GadgetPtr;
    detailPen * : SHORTINT;
    blockPen * : SHORTINT;
    saveColor0 * : INTEGER;
    barLayer * : g.LayerPtr;
    extData * : e.ADDRESS;
    userData * : e.ADDRESS;
  END;

CONST
  filenameSize * = 30;
  pointerSize * = (1+16+1)*2;
  topazEighty * = 8;
  topazSixty * = 9;

(* Preferences.printerType *)
  customName *= 0; alphaP101  *= 1; brother15XL *= 2;  cbmMps1000 *= 3;
  diab630   *= 4; diabAdvD25 *= 5; diabC150   *= 6;  epson      *= 7;
  epsonJX80  *= 8; okimate20  *= 9; qumeLP20    *= 10; hpLaserjet *= 11;
  hpLaserjetPlus * = 12;

CONST
(* PrinterPort *)
  parallelPrinter * = 0;
```

```
  serialPrinter   * = 1;

(* SerParShks *)
  shakeXon   * = 0;
  shakeRts   * = 1;
  shakeNone  * = 2;
  parityNone * = 4;
  parityEven * = 5;
  parityOdd  * = 6;

TYPE
  Preferences * = STRUCT
    fontHeight * : BYTE;
    printerPort * : SHORTINT; (* PrinterPort *)
    baudRate * : INTEGER;
    keyRptSpeed * : Timer.TimeVal;
    keyRptDelay * : Timer.TimeVal;
    doubleClick * : Timer.TimeVal;
    pointerMatrix * : ARRAY pointerSize OF INTEGER;
    xOffset * : SHORTINT;
    yOffset * : SHORTINT;
    color17 * : INTEGER;
    color18 * : INTEGER;
    color19 * : INTEGER;
    pointerTicks * : INTEGER;
    color0 * : INTEGER;
    color1 * : INTEGER;
    color2 * : INTEGER;
    color3 * : INTEGER;
    viewXOffset * : SHORTINT;
    viewYOffset * : SHORTINT;
    viewInitX * : INTEGER;
    viewInitY * : INTEGER;
    enableCLI * : INTEGER;   (* BOOLEAN *)
    printerType * : INTEGER;
    printerFilename * : ARRAY filenameSize OF CHAR;
    printPitch * : INTEGER;
    printQuality * : INTEGER;
    printSpacing * : INTEGER;
    printLeftMargin * : INTEGER;
    printRightMargin * : INTEGER;
    printImage * : INTEGER;
    printAspect * : INTEGER;
    printShade * : INTEGER;
    printThreshold * : INTEGER;
    paperSize * : INTEGER;
    paperLength * : INTEGER;
    paperType * : INTEGER;
```

```
    serRWBits * : BYTE;
    serStopBuf * : BYTE;
    serParShk * : SHORTSET; (* SerParShks *)
    laceWB * : BOOLEAN;
    workName * : ARRAY filenameSize OF CHAR;
    padding * : ARRAY 16 OF SHORTINT;
  END;

CONST
  baud110 * = 0;
  baud300 * = 1;
  baud1200 * = 2;
  baud2400 * = 3;
  baud4800 * = 4;
  baud9600 * = 5;
  baud19200 * = 6;
  baudMidi * = 7;
  pica * = 0H;
  elite * = 0400H;
  fine * = 0800H;
  draft * = 0H;
  letter * = 0100H;
  sixLPI * = 0H;
  eightLPI * = 0200H;
  imagePositive * = 0;
  imageNegative * = 1;
  aspectHoriz * = 0;
  aspectVert * = 1;
  shadeBW * = 0;
  shadeGreyscale * = 1;
  shadeColor * = 2;
  usLetter * = 0H;
  usLegal * = 010H;
  nTractor * = 020H;
  wTractor * = 030H;
  custom * = 040H;
  fanfold * = 0H;
  single * = 080H;
  readBits * = 0F0H;
  writeBits * = 0FH;
  stopBits * = 0F0H;
  bufSizeBits * = 0FH;
  buf512 * = 0;
  buf1024 * = 1;
  buf2048 * = 2;
  buf4096 * = 3;
  buf8000 * = 4;
  buf16000 * = 5;
```

```
TYPE
  Remember * = STRUCT
    nextRemember * : RememberPtr;
    rememberSize * : LONGINT;
    memory * : e.ADDRESS;
  END;

CONST
  deadendAlert * = MIN(LONGINT); recoveryAlert * = 0;

(* DisplayMode *)
  hiresPick  * = 0;
  lowresPick * = 1;
  dModeCount * = 2;

  eventMax * = 10;

(* Res *)
  hiresGadget  * = 0;
  lowresGadget * = 1;

  resCount     * = 2;

(* Gadgets *)
  upFrontGadget   * = 0;
  downBackGadget  * = 1;
  sizeGadget      * = 2;
  closeGadget     * = 3;
  dragGadget      * = 4;
  sUpFrontGadget  * = 5;
  sDownBackGadget * = 6;
  sDragGadget     * = 7;

  gadgetCount     * = 8;

(* ILocks *)
  iStateLock    * = 0;
  layerInfoLock * = 1;
  gadgetsLock   * = 2;
  layerRomLock  * = 3;
  viewLock      * = 4;
  iBaseLock     * = 5;
  rpLock        * = 6;

  numILocks     * = 7;

TYPE
```

```
  FatIntuiMessage * = STRUCT
    intuiMessage * : IntuiMessage;
    prevKeys * : LONGINT;
  END;
  IBox * = STRUCT
    left * : INTEGER;
    top * : INTEGER;
    width * : INTEGER;
    height * : INTEGER;
  END;
  Point * = STRUCT
    x * : INTEGER;
    y * : INTEGER;
  END;
  PenPair * = STRUCT
    detailPen * : SHORTINT;
    blockPen * : SHORTINT;
  END;
  GListEnv * = STRUCT
    screen * : ScreenPtr;
    window * : WindowPtr;
    requester * : RequesterPtr;
    rastPort * : g.RastPortPtr;
    layer * : g.LayerPtr;
    gzzLayer * : g.LayerPtr;
    pens * : PenPair;
    domain * : IBox;
    gzzDims * : IBox;
  END;
  GListEnvPtr * = POINTER TO GListEnv;
  GadgetInfo * = STRUCT
    environ * : GListEnvPtr;
    gadget * : GadgetPtr;
    box * : IBox;
    container * : IBox;
    layer * : g.LayerPtr;
    newKnob * : IBox;
  END;

CONST
  numIEvents * = 4;

TYPE
  IntuitionBase * = STRUCT
    libNode * : e.Library;
    viewLord * : g.View;
    activeWindow * : WindowPtr;
    activeScreen * : ScreenPtr;
```

```
firstScreen * : ScreenPtr;
flags * : LONGSET;
mouseY * : INTEGER;
mouseX * : INTEGER;
time * : Timer.TimeVal;
minXMouse * : INTEGER;
maxXMouse * : INTEGER;
minYMouse * : INTEGER;
maxYMouse * : INTEGER;
startSecs * : LONGINT;
startMicros * : LONGINT;
sysBase * : e.ADDRESS;
gfxBase * : g.GfxBasePtr;
layersBase * : e.ADDRESS;
consoleDevice * : e.ADDRESS;
aPointer * : e.ADDRESS;
aPtrHeight * : BYTE;
aPtrWidth * : SHORTINT;
aXOffset * : SHORTINT;
aYOffset * : SHORTINT;
menuDrawn * : INTEGER;
menuSelected * : INTEGER;
optionList * : INTEGER;
menuRPort * : g.RastPort;
menuTmpRas * : g.TmpRas;
itemCRect * : g.ClipRect;
subCRect * : g.ClipRect;
iBitMap * : g.BitMap;
sBitMap * : g.BitMap;
inputRequest * : e.IOStdReq;
inputInterrupt * : e.Interrupt;
eventKey * : RememberPtr;
iEvents * : e.ADDRESS;
eventCount * : INTEGER;
ieBuffer * : ARRAY numIEvents OF ie.InputEvent;
activeGadget * : GadgetPtr;
activePInfo * : PropInfoPtr;
activeImage * : ImagePtr;
gadgetEnv * : GListEnv;
gadgetInfo * : GadgetInfo;
knobOffset * : Point;
getOKWindow * : WindowPtr;
getOKMessage * : IntuiMessagePtr;
setWExcept * : INTEGER;
gadgetReturn * : INTEGER;
stateReturn * : INTEGER;
rp * : g.RastPortPtr;
```

```
iTmpRas * : g.TmpRas;
oldClipRegion * : g.RegionPtr;
oldScroll * : Point;
iFrame * : IBox;
hthick * : INTEGER;
vthick * : INTEGER;
frameChange * : PROCEDURE();
sizeDrag * : PROCEDURE();
firstPt * : Point;
oldPt * : Point;
sysGadgets * : ARRAY resCount, gadgetCount OF GadgetPtr;
checkImage * : ARRAY resCount OF ImagePtr;
amigaIcon * : ARRAY resCount OF ImagePtr;
aPattern * : ARRAY 8 OF INTEGER;
bPattern * : ARRAY 4 OF INTEGER;
iPointer * : e.ADDRESS;
iPtrHeight * : BYTE;
iPtrWidth * : SHORTINT;
iXOffset * : SHORTINT;
iYOffset * : SHORTINT;
doubleSeconds * : LONGINT;
doubleMicros * : LONGINT;
wBorLeft * ,
wBorTop * ,
wBorRight * ,
wBorBottom * ,
barVBorder * ,
barHBorder * ,
menuVBorder * ,
menuHBorder * : ARRAY dModeCount OF SHORTINT;
color0 * : INTEGER;
color1 * : INTEGER;
color2 * : INTEGER;
color3 * : INTEGER;
color17 * : INTEGER;
color18 * : INTEGER;
color19 * : INTEGER;
sysFont * : g.TextAttr;
preferences * : PreferencesPtr;
echoes * : e.ADDRESS;
viewInitX * : INTEGER;
viewInitY * : INTEGER;
cursorDX * : INTEGER;
cursorDY * : INTEGER;
keyMap * : KeyMap.KeyMapPtr;
mouseYMinimum * : INTEGER;
errorX * : INTEGER;
errorY * : INTEGER;
```

```
    ioExcess * : Timer.IOTimer;
    holdMinYMouse * : INTEGER;
    wbPort * : e.MsgPortPtr;
    fnkuhdPort * : e.MsgPortPtr;
    wbMessage * : IntuiMessage;
    hitScreen * : ScreenPtr;
    simpleSprite * : g.SimpleSpritePtr;
    attachedSSprite * : g.SimpleSpritePtr;
    gotSpritel * : BOOLEAN;
    semaphoreList * : e.List;
    iSemaphore * : ARRAY numlLocks OF e.SignalSemaphore;
    maxDisplayHeight * : INTEGER;
    maxDisplayRow * : INTEGER;
    reserved * : ARRAY 8 OF LONGINT;
  END;
  IntuitionBasePtr * = POINTER TO IntuitionBase;

VAR
  int * :  IntuitionBasePtr;


(* Boolesche Parameter müssen 4 Bytes lang sein: *)

TYPE
  LONGBOOL * = LONGiNT;

CONST
  LTRUE  * = -1;
  LFALSE * = 0;


(*-----------------------------------------------------------------------*)


(*------ intuition.library: ------*)


PROCEDURE ActivateGadget * {int,-462}(gadget{8}:GadgetPtr;
                                      window{9}:WindowPtr;
                                      requester{10}:RequesterPtr):BOOLEAN;
PROCEDURE ActivateWindow * {int,-450}(window{8}:WindowPtr);
PROCEDURE AddGadget * {int,- 42}(window{8}:WindowPtr;
                                 gadget{9}:GadgetPtr;
                                 position{0}:LONGINT):INTEGER;
PROCEDURE AddGList * {int,-438}(window{8}:WindowPtr;
                                gadget{9}:GadgetPtr;
                                position{0}:LONGINT;
                                numGad{1}:LONGINT;
```

```
                    requester{10}:RequesterPtr):INTEGER;
PROCEDURE AllocRemember * {int,-396}(
            VAR rememberKey{8}:e.ADDRESS;
              size{0}:LONGINT;
              flags{1}:LONGSET (* e.MemReqs *)):e.ADDRESS;
PROCEDURE AlohaWorkbench * {int,-402}(wbPort{8}:e.MsgPortPtr);
PROCEDURE AutoRequest * {int,-348}(
                window{8}:WindowPtr;
                bodyText{9}:IntuiTextPtr;
                positiveText{10}:IntuiTextPtr;
                negativeText{11}:IntuiTextPtr;
                positiveFlags{0}:LONGSET; (* IDCMPFlags *)
                negativeFlags{1}:LONGSET; (* IDCMPFlags *)
                width{2}:LONGINT;
                height{3}:LONGINT):BOOLEAN;
PROCEDURE BeginRefresh * {int,-354}(window{8}:WindowPtr);
PROCEDURE BuildSysRequest * {int,-360}(
              window{8}:WindowPtr;
              bodyText{9}:IntuiTextPtr;
              positiveText{10}:IntuiTextPtr;
              negativeText{11}:IntuiTextPtr;
              idcmpFlags{0}:LONGSET; (* IDCMPFlags *)
              width{1}:LONGINT;
              height{2}:LONGINT):WindowPtr;
PROCEDURE ClearDMRequest * {int,- 48}(window{8}:WindowPtr):BOOLEAN;
PROCEDURE ClearMenuStrip * {int,- 54}(window{8}:WindowPtr);
PROCEDURE ClearPointer * {int,- 60}(window{8}:WindowPtr);
PROCEDURE CloseScreen * {int,- 66}(screen{8}:ScreenPtr);
PROCEDURE CloseWindow * {int,- 72}(window{8}:WindowPtr);
PROCEDURE CloseWorkBench * {int,- 78}():BOOLEAN;
PROCEDURE CurrentTime * {int,- 84}(VAR seconds{8}: LONGINT;
                        VAR micros {9}: LONGINT);
PROCEDURE DisplayAlert * {int,- 90}(alertNumber{0}:LONGINT;
                         string{8}:ARRAY OF CHAR;
                         height{1}:LONGINT):BOOLEAN;
PROCEDURE DisplayBeep * {int,- 96}(screen{8}:ScreenPtr);
PROCEDURE DoubleClick * {int,-102}(startSecs{0}:LONGINT;
                        startMicros{1}:LONGINT;
                        currentSecs{2}:LONGINT;
                        currentMicros{3}:LONGINT):BOOLEAN;
PROCEDURE DrawBorder * {int,-108}(rastPort{8}:g.RastPortPtr;
                       border{9}:BorderPtr;
                       leftOffset{0}:LONGINT;
                       topOffset{1}:LONGINT);
PROCEDURE DrawImage * {int,-114}(rastPort{8}:g.RastPortPtr;
                      image{9}:ImagePtr;
                      leftOffset{0}:LONGINT;
                      topOffset{1}:LONGINT);
```

```
PROCEDURE EndRefresh * {int,-366}(window{8}:WindowPtr;
                          complete{0}:LONGBOOL);
PROCEDURE EndRequest * {int,-120}(requester{8}:RequesterPtr;
                          window{9}:WindowPtr);
PROCEDURE FreeRemember * {int,-408}(rememberKey{8}:e.ADDRESS;
                           reallyForget{0}:LONGBOOL);
PROCEDURE FreeSysRequest * {int,-372}(window{8}:WindowPtr);
PROCEDURE GetDefPrefs * {int,-126}(prefBuffer{8}:PreferencesPtr;
                           size{0}:LONGINT);
PROCEDURE GetPrefs * {int,-132}(prefBuffer{8}:PreferencesPtr;
                          size{0}:LONGINT);
PROCEDURE GetScreenData * {int,-426}(buffer{8}:e.ADDRESS;
                             size{0}:LONGINT;
                             type{1}:SET; (* ScreenFlags *)
                             screen{9}:ScreenPtr):BOOLEAN;
PROCEDURE InitRequester * {int,-138}(requester{8}:RequesterPtr);
PROCEDURE IntuiTextLength * {int,-330}(iText{8}:IntuiTextPtr):INTEGER;
PROCEDURE Intuition * {int,- 36}(inputEvent{8}:ie.InputEventPtr);
PROCEDURE ItemAddress * {int,-144}(menuStrip{8}:MenuPtr;
                           menuNumber{0}:LONGINT):MenuItemPtr;
PROCEDURE LockIBase * {int,-414}(lockNumber{0}:LONGINT):LONGINT;
PROCEDURE MakeScreen * {int,-378}(screen{8}:ScreenPtr);
PROCEDURE ModifyIDCMP * {int,-150}(window{8}:WindowPtr;
                           idcmpFlags{0}:LONGSET (* IDCMPFlags *));
PROCEDURE ModifyProp * {int,-156}(gadget{8}:GadgetPtr;
                          window{9}:WindowPtr;
                          requester{10}:RequesterPtr;
                          flags{0}:SET; (* PropInfoFlags *)
                          horizPot{1}:LONGINT;
                          vertPot{2}:LONGINT;
                          horizBody{3}:LONGINT;
                          vertBody{4}:LONGINT);
PROCEDURE MoveScreen * {int,-162}(screen{8}:ScreenPtr;
                          deltaX{0}:LONGINT;
                          deltaY{1}:LONGINT);
PROCEDURE MoveWindow * {int,-168}(window{8}:WindowPtr;
                          deltaX{0}:LONGINT;
                          deltaY{1}:LONGINT);
PROCEDURE NewModifyProp * {int,-468}(gadget{8}:GadgetPtr;
                             window{9}:WindowPtr;
                             requester{10}:RequesterPtr;
                             flags{0}:SET; (* PropInfoFlags *)
                             horizPot{1}:LONGINT;
                             vertPot{2}:LONGINT;
                             horizBody{3}:LONGINT;
                             vertBody{4}:LONGINT;
                             numGad{5}:LONGINT);
PROCEDURE OffGadget * {int,-174}(gadget{8}:GadgetPtr;
```

```
                    window{9}:WindowPtr;
                    requester{10}:RequesterPtr);
PROCEDURE OffMenu * {int,-180}(window{8}:WindowPtr;
                   menuNumber{0}:LONGINT);
PROCEDURE OnGadget * {int,-186}(gadget{8}:GadgetPtr;
                   window{9}:WindowPtr;
                   requester{10}:RequesterPtr);
PROCEDURE OnMenu * {int,-192}(window{8}:WindowPtr;
                  menuNumber{0}:LONGINT);
PROCEDURE OpenIntuition * {int,- 30}():IntuitionBasePtr;
PROCEDURE OpenScreen * {int,-198}(VAR newScreen{8}:NewScreen):ScreenPtr;
PROCEDURE OpenWindow * {int,-204}(
              VAR newWindow{8}:NewWindow):WindowPtr;
PROCEDURE OpenWorkBench * {int,-210}():ScreenPtr;
PROCEDURE PrintIText * {int,-216}(rastPort{8}:g.RastPortPtr;
                    iText{9}:IntuiTextPtr;
                    leftOffset{0}:LONGINT;
                    topOffset{1}:LONGINT);
PROCEDURE RefreshGadgets * {int,-222}(gadgets{8}:GadgetPtr;
                      window{9}:WindowPtr;
                      requester{10}:RequesterPtr);
PROCEDURE RefreshGList * {int,-432}(gadgets{8}:GadgetPtr;
                     window{9}:WindowPtr;
                     requester{10}:RequesterPtr;
                     numGad{0}:LONGINT);
PROCEDURE RefreshWindowFrame * {int,-456}(window{8}:WindowPtr);
PROCEDURE RemakeDisplay * {int,-384}();
PROCEDURE RemoveGadget * {int,-228}(window{8}:WindowPtr;
                     gadget{9}:GadgetPtr):INTEGER;
PROCEDURE RemoveGList * {int,-444}(window{8}:WindowPtr;
                    gadget{9}:GadgetPtr;
                    numgad{0}:LONGINT):INTEGER;
PROCEDURE ReportMouse * {int,-234}(window{8}:WindowPtr;
                    boolean{0}:LONGBOOL);
PROCEDURE Request * {int,-240}(requester{8}:RequesterPtr;
                   window{9}:WindowPtr):BOOLEAN;
PROCEDURE RethinkDisplay * {int,-390}();
PROCEDURE ScreenToBack * {int,-246}(screen{8}:ScreenPtr);
PROCEDURE ScreenToFront * {int,-252}(screen{8}:ScreenPtr);
PROCEDURE SetDMRequest * {int,-258}(window{8}:WindowPtr;
                     dmRequester{9}:RequesterPtr):BOOLEAN;
PROCEDURE SetMenuStrip * {int,-264}(window{8}:WindowPtr;
                     menu{9}:MenuPtr):BOOLEAN;
PROCEDURE SetPointer * {int,-270}(window{8}:WindowPtr;
                    pointer{9}:e.ADDRESS;
                    height{0}:LONGINT;
                    width{1}:LONGINT;
                    xOffset{2}:LONGINT;
```

```
                                yOffset{3}:LONGINT);
PROCEDURE SetPrefs * {int,-324}(prefBuffer{8}:PreferencesPtr;
                               Size{0}:LONGINT;
                               Inform{1}:LONGBOOL);
PROCEDURE SetWindowTitles * {int,-276}(window{8}:WindowPtr;
                                    windowTitle{9}:e.ADDRESS;
                                    screenTitle{10}:e.ADDRESS);
PROCEDURE ShowTitle * {int,-282}(screen{8}:ScreenPtr;
                              showIt{0}:LONGBOOL);
PROCEDURE SizeWindow * {int,-288}(window{8}:WindowPtr;
                               deltaX{0}:LONGINT;
                               deltaY{1}:LONGINT);
PROCEDURE UnlockIBase * {int,-420}(lock{8}:LONGINT);
PROCEDURE ViewAddress * {int,-294}():g.ViewPtr;
PROCEDURE ViewPortAddress * {int,-300}(window{8}:WindowPtr):g.ViewPortPtr;
PROCEDURE WBenchToBack * {int,-336}():BOOLEAN;
PROCEDURE WBenchToFront * {int,-342}():BOOLEAN;
PROCEDURE WindowLimits * {int,-318}(window{8}:WindowPtr;
                                 minWidth{0}:LONGINT;
                                 minHeight{1}:LONGINT;
                                 maxWidth{2}:LONGINT;
                                 maxHeight{3}:LONGINT):BOOLEAN;
PROCEDURE WindowToBack * {int,-306}(window{8}:WindowPtr);
PROCEDURE WindowToFront * {int,-312}(window{8}:WindowPtr);

(* $OvflChk- $RangeChk- $StackChk- $NilChk- $ReturnChk- $CaseChk- *)


(*-----------------------------------------------------------------------*)


(*------ Menu Macros: ------*)


PROCEDURE MenuNum*(n{0}: INTEGER): INTEGER;
BEGIN RETURN s.VAL(INTEGER,     s.VAL(SET,n)     * {0..4}) END MenuNum;

PROCEDURE ItemNum*(n{0}: INTEGER): INTEGER;
BEGIN RETURN s.VAL(INTEGER,s.LSH(s.VAL(SET,n),- 5) * {0..5}) END ItemNum;

PROCEDURE SubNum* (n{0}: INTEGER): INTEGER;
BEGIN RETURN s.VAL(INTEGER,s.LSH(s.VAL(SET,n),-11) * {0..4}) END SubNum;


PROCEDURE ShiftMenu*(n{0}: INTEGER): INTEGER;
BEGIN RETURN s.VAL(INTEGER,     s.VAL(SET,n) * {0..4}    ) END ShiftMenu;

PROCEDURE ShiftItem*(n{0}: INTEGER): INTEGER;
```

```
BEGIN RETURN s.VAL(INTEGER,s.LSH(s.VAL(SET,n) * {0..5}, 5)) END ShiftItem;

PROCEDURE ShiftSub *(n{0}: INTEGER): INTEGER;
BEGIN RETURN s.VAL(INTEGER,s.LSH(s.VAL(SET,n) * {0..4},11)) END ShiftSub;


PROCEDURE MenuNumber*(menu, item, sub: INTEGER): INTEGER;
BEGIN RETURN ShiftMenu(menu) + ShiftItem(item) + ShiftSub(sub)
END MenuNumber;


(*------ Special: ------*)

(* um BOOLEAN-Werte in LONGBOOL umzuwandeln, um sie als bool'schen
   Parameter den Intuition-Prozeduren zu übergeben: *)

PROCEDURE BoolToLong*(b: BOOLEAN): LONGBOOL;
BEGIN IF b THEN RETURN LTRUE
          ELSE RETURN LFALSE END
END BoolToLong;

(* Umwandlung von pseudo unsigned integers (wie z.B. die INTEGERs in
   PropInfo) in LONGINTs: *)

PROCEDURE UIntToLong*(i: INTEGER): LONGINT;
BEGIN
  IF i<0 THEN RETURN i+10000H
        ELSE RETURN i       END;
END UIntToLong;

PROCEDURE LongToUInt*(l: LONGINT): INTEGER;
BEGIN
(* $RangeChk- Trick: einfach unteres Wort zurückgeben *)
  RETURN SHORT(l)
(* $RangeChk= *)
END LongToUInt;

(*-----------------------------------------------------------------------*)

BEGIN
 int := s.VAL(IntuitionBasePtr,e.OpenLibrary("intuition.library",33));
 IF int = NIL THEN HALT(0) END;
CLOSE
 IF int#NIL THEN e.CloseLibrary(s.VAL(e.LibraryPtr,int)) END;
END Intuition.
```

KeyBoard:

```
MODULE Keyboard; (* $Implementation- *)

IMPORT e: Exec;

CONST
  keyboardName * = "keyboard.device";
  readEvent * = e.nonstd+0;
  readMatrix * = e.nonstd+1;
  addResetHandler * = e.nonstd+2;
  remResetHandler * = e.nonstd+3;
  resetHandlerDone * = e.nonstd+4;

END Keyboard.
```

KeyMap:

```
MODULE KeyMap; (* $Implementation- *)

IMPORT e: Exec,
       s: SYSTEM;

CONST
(* KeyMapTypes *)
  shift   * = 0;
  alt     * = 1;
  control * = 2;
  downup  * = 3;
  dead    * = 5;
  string  * = 6;
  nop     * = 7;

  noQual * = SHORTSET{};
  vanilla * = SHORTSET{shift,alt,control};

(* DeadPrefixBytes *)
  dpbMod  * = 0;
  dpbDead * = 3;

  dp2dIndexMask * = 0FH;
  dp2dFacShift * = 4;
  maxKeys * = 64;

TYPE
  BitTable * =  ARRAY maxKeys DIV (8*s.SIZE(LONGSET)) OF LONGSET;
  BitTablePtr * = POINTER TO BitTable;
  KeyInfo * = ARRAY 4 OF CHAR;
  KeyInfoAdr * = e.ADDRESS;
  Types * = ARRAY maxKeys OF SHORTSET; (* KeyMapTypes *)
  TypesPtr * = POINTER TO Types;
  Info * = ARRAY maxKeys OF KeyInfo;
  InfoPtr * = POINTER TO Info;
  KeyMap * = STRUCT
    loKeyMapTypes * : TypesPtr;
    loKeyMap * : InfoPtr;
    loCapsable * : BitTablePtr;
    loRepeatable * : BitTablePtr;
    hiKeyMapTypes * : TypesPtr;
    hiKeyMap * : InfoPtr;
    hiCapsable * : BitTablePtr;
    hiRepeatable * : BitTablePtr;
  END;
```

```
  KeyMapPtr * = POINTER TO KeyMap;
  KeyMapNode * = STRUCT
    node * : e.Node;
    keyMap * : KeyMap;
  END;
  KeyMapResource * = STRUCT
    node * : e.Node;
    list * : e.List;
  END;

END KeyMap.
```

Layers:

```
MODULE Layers;

IMPORT g: Graphics,
       e: Exec,
       s: SYSTEM;

VAR
  base * : e.LibraryPtr;

PROCEDURE BeginUpdate* {base, -78}(l{8}:g.LayerPtr):BOOLEAN;
PROCEDURE BehindLayer* {base, -54}(l{9}:g.LayerPtr):BOOLEAN;
PROCEDURE CreateBehindLayer* {base, -42}(li{8}:g.LayerInfoPtr;
                                bm{9}:g.BitMapPtr;
                                x0{0}:LONGINT;
                                y0{1}:LONGINT;
                                x1{2}:LONGINT;
                                y1{3}:LONGINT;
                                flags{4}:SET; (* g.LayerFlags *)
                                bm2{10}:g.BitMapPtr):g.LayerPtr;
PROCEDURE CreateUpfrontLayer* {base, -36}(li{8}:g.LayerInfoPtr;
                                 bm{9}:g.BitMapPtr;
                                 x0{0}:LONGINT;
                                 y0{1}:LONGINT;
                                 x1{2}:LONGINT;
                                 y1{3}:LONGINT;
                                 flags{4}:SET; (* g.LayerFlags *)
                                 bm2{10}:g.BitMapPtr):g.LayerPtr;
PROCEDURE DeleteLayer* {base, -90}(l{9}:g.LayerPtr):BOOLEAN;
PROCEDURE DisposeLayerInfo* {base,-150}(li{8}:g.LayerInfoPtr);
PROCEDURE EndUpdate* {base, -84}(l{8}:g.LayerPtr; flag{0}:BOOLEAN);
PROCEDURE FattenLayerInfo* {base,-156}(li{8}:g.LayerInfoPtr);
PROCEDURE InitLayers* {base, -30}(li{8}:g.LayerInfoPtr);
PROCEDURE InstallClipRegion* {base,-174}(l{8}:g.LayerPtr;
                                region{9}:g.RegionPtr):g.RegionPtr;
PROCEDURE LockLayer* {base, -96}(l{9}:g.LayerPtr);
PROCEDURE LockLayerInfo* {base,-120}(li{8}:g.LayerInfoPtr);
PROCEDURE LockLayers* {base,-108}(li{8}:g.LayerInfoPtr);
PROCEDURE MoveLayer* {base, -60}(l{9}:g.LayerPtr;
                           dx{0},dy{1}:LONGINT):BOOLEAN;
PROCEDURE MoveLayerInFrontOf* {base,-168}(
                           l{8},target{9}:g.LayerPtr):BOOLEAN;
PROCEDURE NewLayerInfo* {base,-144}():g.LayerInfoPtr;
PROCEDURE ScrollLayer* {base, -72}(l{9}:g.LayerPtr; dx{0},dy{1}:LONGINT);
PROCEDURE SizeLayer* {base, -66}(l{9}:g.LayerPtr;
                           dx{0},dy{1}:LONGINT):BOOLEAN;
```

```
PROCEDURE SwapBitsRastPortClipRect* {base,-126}(rp{8}:g.RastPortPtr;
                                        cr{9}:g.ClipRectPtr);
PROCEDURE ThinLayerInfo* {base,-162}(li{8}:g.LayerInfoPtr);
PROCEDURE UnlockLayer* {base,-102}(l{8}:g.LayerPtr);
PROCEDURE UnlockLayerInfo* {base,-138}(li{8}:g.LayerInfoPtr);
PROCEDURE UnlockLayers* {base,-114}(li{8}:g.LayerInfoPtr);
PROCEDURE UpfrontLayer* {base, -48}(l{9}:g.LayerPtr):BOOLEAN;
PROCEDURE WhichLayer* {base,-132}(li{8}:g.LayerInfoPtr;
                              x{0},y{1}:LONGINT):g.LayerPtr;

(* $OvflChk- $RangeChk- $StackChk- $NilChk- $ReturnChk- $CaseChk- *)

BEGIN

 base := e.OpenLibrary("layers.library",33);
 IF base=NIL THEN HALT(0) END;

CLOSE

 IF base#NIL THEN e.CloseLibrary(base) END;

END Layers.
```

MathFFP:

```
MODULE MathFFP;

IMPORT e: Exec,
       I: Intuition,
       s: SYSTEM;

CONST
  MathFFPName * = "mathffp.library";

VAR
  base* : e.LibraryPtr;

PROCEDURE Fix  * {base,-30}(x{0}     :REAL   ):LONGINT;
PROCEDURE Flt  * {base,-36}(x{0}     :LONGINT):REAL;
PROCEDURE Cmp  * {base,-42}(x{1},y{0}:REAL   ):LONGINT;
PROCEDURE Tst  * {base,-48}(x{1}     :REAL   ):LONGINT;
PROCEDURE Abs  * {base,-54}(x{0}     :REAL   ):REAL;
PROCEDURE Neg  * {base,-60}(x{0}     :REAL   ):REAL;
PROCEDURE Add  * {base,-66}(x{0},y{1}:REAL   ):REAL;
PROCEDURE Sub  * {base,-72}(x{0},y{1}:REAL   ):REAL;
PROCEDURE Mul  * {base,-78}(x{0},y{1}:REAL   ):REAL;
PROCEDURE Div  * {base,-84}(x{0},y{1}:REAL   ):REAL;
PROCEDURE Floor* {base,-90}(x{0}     :REAL   ):REAL;
PROCEDURE Ceil * {base,-96}(x{0}     :REAL   ):REAL;

(* $OvflChk- $RangeChk- $StackChk- $NilChk- $ReturnChk- $CaseChk- *)

BEGIN

  base := e.OpenLibrary(MathFFPName,33);
  IF base=NIL THEN HALT(0) END;

CLOSE

  IF base#NIL THEN e.CloseLibrary(base) END;

END MathFFP.
```

MathIEEEDoubBas:

```
MODULE MathIEEEDoubBas;

IMPORT e: Exec,
       I: Intuition,
       r: Resources,
       s: SYSTEM;

CONST
  MathIEEEDoubBasName * = "mathieeedoubbas.library";

TYPE
 MathIEEEBase=STRUCT
  libNode:e.Library;
  flags:BYTE;
  reserved1:BYTE;
  m68881:e.ADDRESS;
  sysLib:e.ADDRESS;
  segList:e.BPTR;
  resource:r.MathIEEEResourcePtr;
  taskOpenLib:e.ADDRESS;
  taskCloseLib:e.ADDRESS;
 END;

VAR
  base* : e.LibraryPtr;

PROCEDURE Fix*  {base,- 30}(x{0}     :LONGREAL):LONGINT;
PROCEDURE Flt*  {base,- 36}(x{0}     :LONGINT ):LONGREAL;
PROCEDURE Cmp*  {base,- 42}(x{0},y{2}:LONGREAL):LONGINT;
PROCEDURE Tst*  {base,- 48}(x{0}     :LONGREAL):LONGINT;
PROCEDURE Abs*  {base,- 54}(x{0}     :LONGREAL):LONGREAL;
PROCEDURE Neg*  {base,- 60}(x{0}     :LONGREAL):LONGREAL;
PROCEDURE Add*  {base,- 66}(x{0},y{2}:LONGREAL):LONGREAL;
PROCEDURE Sub*  {base,- 72}(x{0},y{2}:LONGREAL):LONGREAL;
PROCEDURE Mul*  {base,- 78}(x{0},y{2}:LONGREAL):LONGREAL;
PROCEDURE Div*  {base,- 84}(x{0},y{2}:LONGREAL):LONGREAL;
PROCEDURE Floor*{base,- 90}(x{0}     :LONGREAL):LONGREAL;
PROCEDURE Ceil* {base,- 96}(x{0}     :LONGREAL):LONGREAL;

(* $OvflChk- $RangeChk- $StackChk- $NilChk- $ReturnChk- $CaseChk- *)

BEGIN

  base := e.OpenLibrary(MathIEEEDoubBasName,33);
  IF base=NIL THEN
```

```
    s.SETREG(0,I.DisplayAlert(0,
      "x00x64x14missing mathieeedoubnas.library",50));
    HALT(0)
  END;

CLOSE

  IF base#NIL THEN e.CloseLibrary(base) END;

END MathIEEEDoubBas.
```

MathIEEEDoubTrans:

```
MODULE MathIEEEDoubTrans;

IMPORT e: Exec,
       I: Intuition,
       s: SYSTEM;

CONST name * = "mathieeedoubtrans.library";

VAR base * : e.LibraryPtr;

PROCEDURE Acos  * {base,-120}(x{0}:LONGREAL):LONGREAL;
PROCEDURE Asin  * {base,-114}(x{0}:LONGREAL):LONGREAL;
PROCEDURE Atan  * {base,- 30}(x{0}:LONGREAL):LONGREAL;
PROCEDURE Cos   * {base,- 42}(x{0}:LONGREAL):LONGREAL;
PROCEDURE Cosh  * {base,- 66}(x{0}:LONGREAL):LONGREAL;
PROCEDURE Exp   * {base,- 78}(x{0}:LONGREAL):LONGREAL;
PROCEDURE Fieee * {base,-108}(x{0}:REAL    ):LONGREAL;
PROCEDURE Log   * {base,- 84}(x{0}:LONGREAL):LONGREAL;
PROCEDURE Log10 * {base,-126}(x{0}:LONGREAL):LONGREAL;
PROCEDURE Pow   * {base,- 90}(exp{0},x{2}:LONGREAL):LONGREAL;
PROCEDURE Sin   * {base,- 36}(x{0}:LONGREAL):LONGREAL;
PROCEDURE Sincos* {base,- 54}(x{0}:LONGREAL;
                              VAR cos{8}:LONGREAL):LONGREAL;
PROCEDURE Sinh  * {base,- 60}(x{0}:LONGREAL):LONGREAL;
PROCEDURE Sqrt  * {base,- 96}(x{0}:LONGREAL):LONGREAL;
PROCEDURE Tan   * {base,- 48}(x{0}:LONGREAL):LONGREAL;
PROCEDURE Tanh  * {base,- 72}(x{0}:LONGREAL):LONGREAL;
PROCEDURE Tieee * {base,-102}(x{0}:LONGREAL):REAL;

(* $OvflChk- $RangeChk- $StackChk- $NilChk- $ReturnChk- $CaseChk- *)

BEGIN

  base := e.OpenLibrary(s.ADR(name),34);
  IF base=NIL THEN
    s.SETREG(0,I.DisplayAlert(0,"x00x64x14missing mathieeedoubtrans.library",50));
    HALT(0)
  END;

CLOSE

  IF base#NIL THEN e.CloseLibrary(base) END;

END MathIEEEDoubTrans.
```

MathTrans:

```
MODULE MathTrans;

IMPORT e:  Exec,
       I:  Intuition,
       s:  SYSTEM;

CONST MathTransName * = "mathtrans.library";

VAR
  base * : e.LibraryPtr;

PROCEDURE Acos  * {base,-120}(x{0}  :REAL):REAL;
PROCEDURE Asin  * {base,-114}(x{0}  :REAL):REAL;
PROCEDURE Atan  * {base,- 30}(x{0}  :REAL):REAL;
PROCEDURE Cos   * {base,- 42}(x{0}  :REAL):REAL;
PROCEDURE Cosh  * {base,- 66}(x{0}  :REAL):REAL;
PROCEDURE Exp   * {base,- 78}(x{0}  :REAL):REAL;
PROCEDURE Fieee * {base,-108}(x{0}  :LONGINT):REAL;
PROCEDURE Log   * {base,- 84}(x{0}  :REAL):REAL;
PROCEDURE Log10 * {base,-126}(x{0}  :REAL):REAL;
PROCEDURE Pow   * {base,- 90}(e{1},b{0}:REAL):REAL;
PROCEDURE Sin   * {base,- 36}(x{0}  :REAL):REAL;
PROCEDURE Sincos* {base,- 54}(x{0}  :REAL;
                          VAR cos{1}  :REAL):REAL;
PROCEDURE Sinh  * {base,- 60}(x{0}  :REAL):REAL;
PROCEDURE Sqrt  * {base,- 96}(x{0}  :REAL):REAL;
PROCEDURE Tan   * {base,- 48}(x{0}  :REAL):REAL;
PROCEDURE Tanh  * {base,- 72}(x{0}  :REAL):REAL;
PROCEDURE Tieee * {base,-102}(x{0}  :REAL):LONGINT;

(* $OvflChk- $RangeChk- $StackChk- $NilChk- $ReturnChk- $CaseChk- *)

BEGIN

  base := e.OpenLibrary(MathTransName,33);
  IF base=NIL THEN
    s.SETREG(0,I.DisplayAlert(0,"x00x64x14missing mathtrans.library",50));
    HALT(0)
  END;

CLOSE

  IF base#NIL THEN e.CloseLibrary(base) END;

END MathTrans.
```

Narrator:

```
MODULE Narrator; (* $Implementation- *)

IMPORT e: Exec;

CONST
  narratorName * = "narrator.device";
  noMem * = -2;
  noAudLib * = -3;
  makeBad * = -4;
  unitErr * = -5;
  cantAlloc * = -6;
  unimpl * = -7;
  noWrite * = -8;
  expunged * = -9;
  phonErr * = -20;
  rateErr * = -21;
  pitchErr * = -22;
  sexErr * = -23;
  modeErr * = -24;
  freqErr * = -25;
  volErr * = -26;

CONST
  male * = 0;
  female * = 1;
  natural * = 0;
  robotic * = 1;
  defPitch * = 110;
  defRate * = 150;
  defVol * = 64;
  defFreq * = 22200;
  defSex * = male;
  defMode * = natural;
  minRate * = 40;
  maxRate * = 400;
  minPitch * = 65;
  maxPitch * = 320;
  minFreq * = 5000;
  maxFreq * = 28000;
  minVol * = 0;
  maxVol * = 64;

TYPE
  Narrator * = STRUCT
    message * : e.IOStdReq;
```

```
    rate * : INTEGER;
    pitch * : INTEGER;
    mode * : INTEGER;
    sex * : INTEGER;
    chMasks * : e.ADDRESS;
    nmMasks * : INTEGER;
    volume * : INTEGER;
    sampFreq * : INTEGER;
    mouths * : BYTE;
    chanMask * : BYTE;
    numChan * : BYTE;
    pad * : BYTE;
  END;
  NarratorPtr * = POINTER TO Narrator;
(* Synonyme mit einheitlicher Namensgebung *)
  IONarrator * = Narrator;
  IONarratorPtr * = NarratorPtr;
  Mouth * = STRUCT
    voice * : IONarrator;
    width * : BYTE;
    height * : BYTE;
    shape * : BYTE;
    pad * : BYTE;
  END;
  MouthPtr * = POINTER TO Mouth;

END Narrator.
```

Parallel:

```
MODULE Parallel; (* $Implementation- *)

IMPORT e: Exec;

CONST
 parallelName * = "parallel.device";
 query * = e.nonstd;
 setParams * = e.nonstd+1;

TYPE
 IOPArray * = STRUCT
  pTermArray0 * : LONGINT;
  pTermArray1 * : LONGINT;
 END;

CONST
(* ParErr *)
 devBusy * = 1;
 bufTooBig * = 2;
 invParam * = 3;
 lineErr * = 4;
 notOpen * = 5;
 portReset * = 6;
 initErr * = 7;

(* ParFlags *)
 eofMode * = 1;
 radBoogie * = 3;
 shared * = 5;

(* Status *)
 pBusy * = 0;
 paperOut * = 1;
 pSel * = 2;
 rwDir * = 3;
 active * = 4;
 abort * = 5;
 queue * = 6;

TYPE
 IOExtPar * = STRUCT
  ioPar * : e.IOStdReq;
  pExtFlags * : LONGINT;
  status * : SHORTSET;  (* Status *)
  parFlags * : SHORTSET; (* ParFlags *)
```

```
 pTermArray * : IOPArray;
END;
IOExtParPtr * = POINTER TO IOExtPar;

IOParallel * = IOExtPar;
IOParallelPtr * = IOExtParPtr;

END Parallel.
```

Printer:

```
MODULE Printer; (* $Implementation- *)

IMPORT e: Exec,
       g: Graphics;

CONST
 printerName * = "printer.device";
 rawWrite * = e.nonstd+0;
 prtCommand * = e.nonstd+1;
 dumpRPort * = e.nonstd+2;
 query * = e.nonstd+3;

 ris * = 0;          (* ESCc  reset                      ISO *)
 rin * = 1;          (* ESC#1 initialize                 +++ *)
 ind * = 2;          (* ESCD  lf                         ISO *)
 nel * = 3;          (* ESCE  return,lf                  ISO *)
 ri * = 4;           (* ESCM  reverse lf                 ISO *)

 sgr0 * = 5;         (* ESC[0m normal char set           ISO *)
 sgr3 * = 6;         (* ESC[3m italics on                ISO *)
 sgr23 * = 7;        (* ESC[23m italics off              ISO *)
 sgr4 * = 8;         (* ESC[4m underline * = 0;          ISO *)
 sgr24 * = 9;        (* ESC[24m underline off            ISO *)
 sgr1 * = 10;        (* ESC[1m boldface on               ISO *)
 sgr22 * = 11;       (* ESC[22m boldface off             ISO *)
 sfc * = 12;         (* SGR30-39  set foreground color   ISO *)
 sbc * = 13;         (* SGR40-49  set background color   ISO *)

 shorp0 * = 14;      (* ESC[0w normal pitch              DEC *)
 shorp2 * = 15;      (* ESC[2w elite on                  DEC *)
 shorp1 * = 16;      (* ESC[1w elite off                 DEC *)
 shorp4 * = 17;      (* ESC[4w condensed fine on         DEC *)
 shorp3 * = 18;      (* ESC[3w condensed off             DEC *)
 shorp6 * = 19;      (* ESC[6w enlarged on               DEC *)
 shorp5 * = 20;      (* ESC[5w enlarged off              DEC *)

 den6 * =  21;       (* ESC[6"z shadow print on          DEC (sort of) *)
 den5 * =  22;       (* ESC[5"z shadow print off         DEC *)
 den4 * =  23;       (* ESC[4"z doublestrike on          DEC *)
 den3 * =  24;       (* ESC[3"z doublestrike off         DEC *)
 den2 * =  25;       (* ESC[2"z  NLQ on                  DEC *)
 den1 * =  26;       (* ESC[1"z  NLQ off                 DEC *)

 sus2 * =  27;       (* ESC[2v superscript on            +++ *)
 sus1 * =  28;       (* ESC[1v superscript off           +++ *)
```

```
sus4 * =  29;     (* ESC[4v subscript on                  +++ *)
sus3 * =  30;     (* ESC[3v subscript off                 +++ *)
sus0 * =  31;     (* ESC[0v normalize the line            +++ *)
plu * =  32;      (* ESCL  partial line up                ISO *)
pld * =  33;      (* ESCK  partial line down              ISO *)

fnt0 * =  34;     (* ESC(B US char set                    DEC *)
fnt1 * =  35;     (* ESC(R French char set                DEC *)
fnt2 * =  36;     (* ESC(K German char set                DEC *)
fnt3 * =  37;     (* ESC(A UK char set                    DEC *)
fnt4 * =  38;     (* ESC(E Danish I char set              DEC *)
fnt5 * =  39;     (* ESC(H Sweden char set                DEC *)
fnt6 * =  40;     (* ESC(Y Italian char set               DEC *)
fnt7 * =  41;     (* ESC(Z Spanish char set               DEC *)
fnt8 * =  42;     (* ESC(J Japanese char set              +++ *)
fnt9 * =  43;     (* ESC(6 Norweign char set              DEC *)
fnt10 * = 44;     (* ESC(C Danish II char set             +++ *)

prop2 * = 45;     (* ESC[2p  proportional on              +++ *)
prop1 * = 46;     (* ESC[1p  proportional off             +++ *)
prop0 * = 47;     (* ESC[0p  proportional clear           +++ *)
tss * =  48;      (* ESC[n E set proportional offset      ISO *)
jfy5 * =  49;     (* ESC[5 F auto left justify            ISO *)
jfy7 * =  50;     (* ESC[7 F auto right justify           ISO *)
jfy6 * =  51;     (* ESC[6 F auto full justify            ISO *)
jfy0 * =  52;     (* ESC[0 F auto justify off             ISO *)
jfy3 * =  53;     (* ESC[3 F letter space (justify)       ISO (special) *)
jfy1 * =  54;     (* ESC[1 F word fill(auto center)       ISO (special) *)

verp0 * = 55;     (* ESC[0z  1/8" line spacing            +++ *)
verp1 * = 56;     (* ESC[1z  1/6" line spacing            +++ *)
slpp * =  57;     (* ESC[nt  set form length n            DEC *)
perf * =  58;     (* ESC[nq  perf skip n (n>0)            +++ *)
perf0 * = 59;     (* ESC[0q  perf skip off                +++ *)

lms * =  60;      (* ESC#9  Left margin set               +++ *)
rms * =  61;      (* ESC#0  Right margin set              +++ *)
tms * =  62;      (* ESC#8  Top margin set                +++ *)
bms * =  63;      (* ESC#2  Bottom marg set               +++ *)
stbm * =  64;     (* ESC[Pn1;Pn2r  T&B margins            DEC *)
slrm * =  65;     (* ESC[Pn1;Pn2s  L&R margin             DEC *)
cam * =  66;      (* ESC#3  Clear margins                 +++ *)

hts * =  67;      (* ESCH   Set horiz tab                 ISO *)
vts * =  68;      (* ESCJ   Set vertical tabs             ISO *)
tbc0 * =  69;     (* ESC[0g  Clr horiz tab                ISO *)
tbc3 * =  70;     (* ESC[3g  Clear all h tab              ISO *)
tbc1 * =  71;     (* ESC[1g  Clr vertical tabs            ISO *)
```

```
 tbc4 * =  72;        (* ESC[4g  Clr all v tabs                ISO *)
 tbcall * = 73;       (* ESC#4   Clr all h & v tabs            +++ *)
 tbsall * = 74;       (* ESC#5   Set default tabs              +++ *)
 extend * = 75;       (* ESC[Pn"x extended commands            +++ *)
 raw * =   76;        (* ESC[Pn"r next Pn chars are raw        +++ *)

(* Error *)
 noErr * = 0;
 cancel * = 1;
 notGraphics * = 2;
 invertHam * = 3;
 badDimension * = 4;
 dimensionOvflow * = 5;
 internalMemory * = 6;
 bufferMemory * = 7;
 tookControl * = 8;

TYPE
 IOPrtCmdReq * = STRUCT
  message * : e.Message;
  device * : e.DevicePtr;
  unit * : e.UnitPtr;
  command * : INTEGER;
  flags * : SHORTSET;
  error * : SHORTINT; (* Error *)
  prtCommand * : INTEGER;
  parm0 * : BYTE;
  parm1 * : BYTE;
  parm2 * : BYTE;
  parm3 * : BYTE;
 END;
 IOPrtCmdReqPtr * = POINTER TO IOPrtCmdReq;

CONST
(* Special *)
 milCols * = 0;
 milRows * = 1;
 fullCols * = 2;
 fullRows * = 3;
 fracCols * = 4;
 fracRows * = 5;
 center * = 6;
 aspect * = 7;
 density1 * = 8;
 density2 * = 9;
 density4 * = 10;
 noFormFeed * = 11;
 trustMe * = 12;
```

```
 noPrint * = 13;

 density3 * = {density1,density2};
 density5 * = {density1,density4};
 density6 * = {density2,density4};
 density7 * = {density1..density4};
 densityMask * = {density1..density4};
 dimensionMask * = {milCols..fracRows,aspect};

TYPE
 IODRPReq * = STRUCT
  message * : e.Message;
  device * : e.DevicePtr;
  unit * : e.UnitPtr;
  command * : INTEGER;
  flags * : SHORTSET;
  error * : SHORTINT; (* Error *)
  rastPort * : g.RastPortPtr;
  colorMap * : g.ColorMapPtr;
  modesHi * : INTEGER; (*  * :  * = 0 *)
  modes * : SET; (* Graphics.ViewModes *)
  srcX * : INTEGER;
  srcY * : INTEGER;
  srcWidth * : INTEGER;
  srcHeight * : INTEGER;
  destCols * : LONGINT;
  destRows * : LONGINT;
  special * : SET; (* Special *)
 END;
 IODRPReqPtr * = POINTER TO IODRPReq;

(* Synonyme mit einheitlicher Namensgebung *)
 IOPrinter * = IOPrtCmdReq;
 IOPrinterPtr * = IOPrtCmdReqPtr;

END Printer.
```

PrtBase:

```
MODULE PrtBase; (* $Implementation- *)

IMPORT e: Exec,
       I: Intuition,
       p: Parallel,
       s: Serial,
       t: Timer;

TYPE
  DeviceData * = STRUCT
    device * : e.Device;
    segment * : e.BPTR;
    execBase * : e.ExecBasePtr;
    cmdVectors * : e.ADDRESS;
    cmdBytes * : e.ADDRESS;
    numCommands * : INTEGER;
  END;
  DeviceDataPtr * = POINTER TO DeviceData;

CONST
  bufSize * = 256;
  safeSize * = 128;
  stkSize * = 0800H;

TYPE
  PrinterSegmentPtr * = POINTER TO PrinterSegment;
  PrinterData * = STRUCT
    device * : DeviceData;
    unit * : e.MsgPort;
    printerSegment * : e.BPTR;
    printerType * : INTEGER;
    segmentData * : PrinterSegmentPtr;
    printBuf * : e.ADDRESS;
    pWrite * : PROCEDURE(): INTEGER;
    pBothReady * : PROCEDURE(): INTEGER;
    p0,p1 * : p.IOParallel; (* oder s.IOSerial *)
    tior * : t.IOTimer;
    iorPort * : e.MsgPort;
    tc * : e.Task;
    stk * : ARRAY stkSize OF BYTE;
    flags * : BYTE;
    pad * : BYTE;
    preferences * : I.Preferences;
    pWaitEnabled * : BYTE;
  END;
```

```
  PrinterDataPtr * = POINTER TO PrinterData;

CONST
(* PrinterClass *)
  gfx * = 0;
  color * = 1;

  bwAlpha * = SHORTSET{};
  bwGfx * = SHORTSET{gfx};
  colorAlpha * = SHORTSET{color};
  colorGfx * = SHORTSET{gfx,color};


(* ColorClass *)
  blackAndWhite * = 0;
  colors * = 1;
  fourColor * = 2;
  additive * = 3;
  multipass * = 4;

  bw * = SHORTSET{blackAndWhite};
  ymc * = SHORTSET{colors};
  ymcBw * = SHORTSET{blackAndWhite,colors};
  ymcb * = SHORTSET{fourColor};
  wb * = SHORTSET{blackAndWhite,additive};
  bgr * = SHORTSET{colors,additive};
  bgrWb * = SHORTSET{blackAndWhite,colors,additive};
  bgrw * = SHORTSET{fourColor,additive};

TYPE
  PrinterExtendedData * = STRUCT
    printerName * : e.ADDRESS;
    init * : PROCEDURE();
    expunge * : PROCEDURE();
    open * : PROCEDURE(): INTEGER;
    close * : PROCEDURE();
    printerClass * : SHORTSET; (* PrinterClass *)
    colorClass * : SHORTSET;  (* ColorClass   *)
    maxColumns * : BYTE;
    numCharSets * : BYTE;
    numRows * : INTEGER;
    maxXDots * : LONGINT;
    maxYDots * : LONGINT;
    xDotsInch * : INTEGER;
    yDotsInch * : INTEGER;
    commands * : e.ADDRESS;
    doSpecial * : PROCEDURE(): LONGINT;
    render * : PROCEDURE(): LONGINT;
```

```
    timeoutSecs * : LONGINT;
    eightBitChars * : e.ADDRESS;
    printMode * : LONGINT;
    convFunc * : PROCEDURE(): LONGINT;
  END;
  PrinterExtendedDataPtr * = POINTER TO PrinterExtendedData;
  PrinterSegment * = STRUCT
    nextSegment * : e.BPTR;
    runAlert * : LONGINT;
    version * : INTEGER;
    revision * : INTEGER;
    ped * : PrinterExtendedData;
  END;

END PrtBase.
```

Resources:

```
MODULE Resources;

IMPORT
  e: Exec;

CONST
  ciaaName * = "ciaa.resource";
  ciabName * = "ciab.resource";

TYPE
  CiaResourcePtr * = e.ADDRESS;

VAR
    cia * :  CiaResourcePtr; (* Setzen, bevor Cia-Routinen benutzt werden! *)


TYPE
  DiscResourceUnit * = STRUCT
    message * : e.Message;
    discBlock * : e.Interrupt;
    discSync * : e.Interrupt;
    index * : e.Interrupt
  END;
  DiscResourceUnitPtr * = POINTER TO DiscResourceUnit;

CONST
(* DiscResourceFlags *)
  alloc0 * = 0;
  alloc1 * = 1;
  alloc2 * = 2;
  alloc3 * = 3;
  active * = 7;

  numAlloc * = 4;

TYPE
  DiscResource * = STRUCT
    library * : e.Library;
    current * : DiscResourceUnitPtr;
    flags * : SHORTSET; (* DiscResourceFlags *)
    pad * : BYTE;
    sysLib * : e.LibraryPtr;
    ciaResource * : e.LibraryPtr;
    unitID * : ARRAY numAlloc OF LONGINT;
    waiting * : e.List;
```

```
    discBlock * : e.Interrupt;
    discSync * : e.Interrupt;
    index * : e.Interrupt;
  END;
  DiscResourcePtr * = POINTER TO DiscResource;

CONST
  diskName * = "disk.resource";
  dskDmaOff * = 4000H;
  amiga * = 0;
  drt37422D2S * = 55555555H;
  empty * = -1;

VAR
  disk * :  DiscResourcePtr; (* Setzen, bevor Disc-Routinen benutzt werden! *)

CONST
  fsrName * = "FileSystem.resource";
TYPE
  FileSysResource * = STRUCT
    node * : e.Node;
    creator * : e.ADDRESS;
    fileSysEntries * : e.List;
  END;
  FileSysEntry * = STRUCT
    node * : e.Node;
    dosType * : LONGINT;
    version * : LONGINT;
    patchFlags * : LONGINT;
    type * : LONGINT;
    task * : e.TaskPtr;
    lock * : e.BPTR; (* Dos.FileLockPtr *)
    handler * : e.BPTR;
    stackSize * : LONGINT;
    priority * : LONGINT;
    startup * : e.BPTR;
    segList * : e.BPTR;
    globalVec * : e.BPTR;
  END;

CONST
(* MathIEEEResourceFlags *)
  dblbas * = 0;
  dbltrans * = 1;
  sglbas * = 2;
  sgltrans * = 3;
  extbas * = 4;
  exttrans * = 5;
```

```
TYPE
  PROC * = PROCEDURE();
  MathIEEEResource * = STRUCT
    node * : e.Node;
    flags * : SET; (* MathIEEEResourceFlags *)
    baseAddr * : e.ADDRESS;
    dblBasInit * : PROC;
    dblTransInit * : PROC;
    sglBasInit * : PROC;
    sglTransInit * : PROC;
    extBasInit * : PROC;
    extTransInit * : PROC;
  END;
  MathIEEEResourcePtr * = POINTER TO MathIEEEResource;

CONST
  miscName * = "misc.resource";

CONST
(* ResourceTypes *)
  serialPort * = 0;
  serialBits * = 1;
  parallelPort * = 2;
  parallelBits * = 3;
  numResourceTypes * = 4;

TYPE
  MiscResource * = STRUCT
    library * : e.Library;
    allocArray * : ARRAY numResourceTypes OF e.ADDRESS;
  END;
  MiscResourcePtr * = POINTER TO MiscResource;

VAR
  misc * : MiscResourcePtr; (* Setzen, bevor Misc-Routinen benutzt werden! *)


CONST
  potgoName * = "potgo.resource";

VAR
  potgo * : e.ADDRESS; (* Setzen, bevor Potgo-Routinen benutzt werden! *)


(* CIA-Prozeduren: (vor Verwendung cia setzen !) *)

PROCEDURE AbleICR * {cia,-18}(mask{0}:SHORTSET (* Hardware.CiaIcrFlags *)
```

```
                    ):SHORTSET; (* Hardware.CiaIcrFlags *)
PROCEDURE AddICRVector * {cia,-6}(
                    icrBit{0}:SHORTSET; (* Hardware.CiaIcrFlags *)
                    interrupt{9}:e.InterruptPtr
                    ):e.InterruptPtr;
PROCEDURE RemICRVector * {cia,-12}(
                    icrBit{0}:SHORTSET; (* Hardware.CiaIcrFlags *)
                    interrupt{9}:e.InterruptPtr);
PROCEDURE SetICR * {cia,-24}(cia{14}:CiaResourcePtr;
                mask{0}:SHORTSET (* Hardware.CiaIcrFlags *)
                ):SHORTSET; (* Hardware.CiaIcrFlags *)


(* DISC-Prozeduren: (vor Verwendung disk setzen !) *)

PROCEDURE AllocUnit * {disk,-6}(unitNum{0}:LONGINT):BOOLEAN;
PROCEDURE FreeUnit * {disk,-12}(unitNum{0}:LONGINT);
PROCEDURE GetUnit * {disk,-18}(unitPointer{9}:DiscResourceUnitPtr
                ):DiscResourceUnitPtr;
PROCEDURE GetUnitID * {disk,-30}(unitNum{0}:LONGINT):LONGINT;
PROCEDURE GiveUnit * {disk,-24};

(* MISC-Prozeduren: (vor Verwendung misc setzen !) *)

PROCEDURE AllocMiscResource * {misc,-6}(unitNum{0}:LONGINT;
                    name{9}: ARRAY OF CHAR):e.ADDRESS;
PROCEDURE FreeMiscResource * {misc,-12}(unitNum{0}:LONGINT);

(* POTGO-Prozeduren: (vor Verwendung potgo setzen !) *)

PROCEDURE AllocPotBits * {potgo,-6}(bits{0}: SET  (* Hardware.PotFlags *)
                    ): SET; (* Hardware.PotFlags *)
PROCEDURE FreePotBits * {potgo,-12}(allocated{0}: SET (* Hardware.PotFlags *));
PROCEDURE WritePotgo * {potgo,-18}(word{0}: SET; (* Hardware.PotFlags *)
                    mask{1}: SET  (* Hardware.PotFlags *));

END Resources.
```

SCSI:

```
MODULE SCSI; (* $Implementation- *)

IMPORT e: Exec;

CONST
  scsiCmd * = 28;

(* SCSIFlags: *)
  write * = 0;
  read * = 1;

TYPE
  SCSICmd * = STRUCT
    data * : e.ADDRESS;
    length * : LONGINT;
    actual * : LONGINT;
    command * : e.ADDRESS;
    cmdLength * : INTEGER;
    cmdActual * : INTEGER;
    flags * : SHORTSET; (* SCSIFlags *)
    status * : BYTE;
  END;
  SCSICmdPtr * = POINTER TO SCSICmd;

CONST
  SelfUnit * = 40;
  DMA * = 41;
  Phase * = 42;
  Parity * = 43;
  SelTimeout * = 44;
  BadStatus * = 45;
  NoBoard * = 50;

END SCSI.
```

Serial:

```
MODULE Serial; (* $Implementation- *)

IMPORT e: Exec;

CONST
  serialName * = "serial.device";
  query * = e.nonstd;
  break * = e.nonstd+1;
  setParams * = e.nonstd+2;
  active * = SHORTSET{4};
  abort * = SHORTSET{5};
  queued * = SHORTSET{6};
  bufrRead * = SHORTSET{7};

TYPE
  IOTArray * = STRUCT
    termArray0 * : LONGINT;
    termArray1 * : LONGINT;
  END;

CONST
(* SerFlags *)
  partyOn * = 0;
  partyOdd * = 1;
  sevenWire * = 2;
  queuedBrk * = 3;
  radBoogie * = 4;
  shared * = 5;
  eofMode * = 6;
  xDisabled * = 7;
  mark * = 8;
  mSpOn * = 9;

(* Status *)
  busy * = 0;
  paperOut * = 1;
  select * = 2;
  dataSetReady * = 3;
  clearToSend * = 4;
  carrierDetect * = 5;
  readyToSend * = 6;
  dataTerminalReady * = 7;
  overrun * = 8;
  wroteBreak * = 9;
  readBreak * = 10;
```

```
  xOffWrite * = 11;
  xOffRead * = 12;

  ringIndicator * = select;

(* Error *)
  devBusy * = 1;
  baudMismatch * = 2;
  invBaud * = 3;
  bufErr * = 4;
  invParam * = 5;
  lineErr * = 6;
  notOpen * = 7;
  portReset * = 8;
  parityErr * = 9;
  initErr * = 10;
  timerErr * = 11;
  bufOverflow * = 12;
  nodsr * = 13;
  nocts * = 14;
  detectedBreak * = 15;

TYPE
  IOExtSer * = STRUCT
    ioSer * : e.IOStdReq;
    ctlChar * : LONGINT;
    rBufLen * : LONGINT;
    extFlags * : LONGSET; (* SerFlags *)
    baud * : LONGINT;
    brkTime * : LONGINT;
    termArray * : IOTArray;
    readLen * : BYTE;
    writeLen * : BYTE;
    stopBits * : BYTE;
    serFlags * : SHORTSET; (* SerFlags (<8!) *)
    status * : SET; (* Status *)
  END;
  IOExtSerPtr * = POINTER TO IOExtSer;

(* Synonyme mit einheitlicher Namesgebung *)
  IOSerial * = IOExtSer;
  IOSerialPtr * = IOExtSerPtr;

END Serial.
```

Timer:

```
MODULE Timer;

IMPORT e: Exec;

CONST
  timerName * = "timer.device";
  addRequest * = e.nonstd+0;
  getSysTime * = e.nonstd+1;
  setSysTime * = e.nonstd+2;
  microHz * = 0;
  vBlank * = 1;

TYPE
  TimeVal * = STRUCT
    secs * : LONGINT;
    micro * : LONGINT
  END;
  TimeValPtr * = POINTER TO TimeVal;
  TimeRequest * = STRUCT
    node * : e.IORequest;
    time * : TimeVal
  END;
  TimeRequestPtr * = POINTER TO TimeRequest;

  IOTimer * = TimeRequest;
  IOTimerPtr * = TimeRequestPtr;

VAR
  timer*: e.DevicePtr; (* muß auf das Timer-Device zeigen! *)

PROCEDURE AddTime * {timer,-42}(VAR dest{8},source{9}: TimeVal);
PROCEDURE CmpTime * {timer,-54}(VAR tv1{8} ,tv2{9}   : TimeVal): INTEGER;
PROCEDURE SubTime * {timer,-48}(VAR dest{8},source{9}: TimeVal);

END Timer.
```

TrackDisk:

```
MODULE TrackDisk; (* $Implementation- *)

IMPORT e: Exec;

CONST
  trackDiskName * = "trackdisk.device";
  motor * = e.nonstd;
  seek * = e.nonstd+1;
  format * = e.nonstd+2;
  remove * = e.nonstd+3;
  changeNum * = e.nonstd+4;
  changeState * = e.nonstd+5;
  protStatus * = e.nonstd+6;
  rawRead * = e.nonstd+7;
  rawWrite * = e.nonstd+8;
  getDriveType * = e.nonstd+9;
  getNumTracks * = e.nonstd+10;
  addChangeInt * = e.nonstd+11;
  remChangeInt * = e.nonstd+12;
  lastComm * = e.nonstd+13;
  extCom * = 8000H;
  extWrite * = e.write+extCom;
  extRead * = e.read+extCom;
  extMotor * = motor+extCom;
  extSeek * = seek+extCom;
  extFormat * = format+extCom;
  extUpdate * = e.update+extCom;
  extClear * = e.clear+extCom;
  extRawRead * = rawRead+extCom;
  extRawWrite * = rawWrite+extCom;
  numSecs * = 11;
  numUnits * = 4;
  sector * = 512;
  secShift * = 9;
  labelSize * = 16;
  indexSync * = 4;
  allowNon35 * = 0;
  drive35 * = 1;
  drive525 * = 2;
  notSpecified * = 20;
  noSecHdr * = 21;
  badSecPreamble * = 22;
  badSecId * = 23;
  badHdrSum * = 24;
  badSecSum * = 25;
```

```
  tooFewSecs * = 26;
  badSecHdr * = 27;
  writeProt * = 28;
  diskChanged * = 29;
  seekError * = 30;
  noMem * = 31;
  badUnitNum * = 32;
  badDriveType * = 33;
  driveInUse * = 34;
  postReset * = 35;

TYPE
  IOExtTD * = STRUCT
    req * : e.IOStdReq;
    count * : LONGINT;
    secLabel * : LONGINT;
  END;
  IOExtTDPtr * = POINTER TO IOExtTD;
  TDUPublicUnit * = STRUCT
    unit * : e.Unit;
    comp01Track * : INTEGER;
    comp10Track * : INTEGER;
    comp11Track * : INTEGER;
    stepDelay * : LONGINT;
    settleDelay * : LONGINT;
    retryCnt * : BYTE;
  END;

(* Synonyme zur einheitlichen Namesgebung *)
  IOTrackDisk * = IOExtTD;
  IOTrackDiskPtr * = IOExtTDPtr;

END TrackDisk.
```

Translator:

```
MODULE Translator;

IMPORT e: Exec,
       I: Intuition,
       s: SYSTEM;

CONST
  notUsed * = -1;
  noMem * = -2;
  makeBad * = -4;

  TranslatorName * = "translator.library";

VAR
  base * : e.LibraryPtr;

PROCEDURE Translate * {base,-30}(in{8}:e.ADDRESS;
                                 inLen{0}:LONGINT;
                                 out{9}:e.ADDRESS;
                                 outLen{1}:LONGINT): LONGINT;

(* $OvflChk- $RangeChk- $StackChk- $NilChk- $ReturnChk- $CaseChk- *)

BEGIN

  base := e.OpenLibrary(TranslatorName,34);
  IF base=NIL THEN
    s.SETREG(0,I.DisplayAlert(0,"\x00\x64\x14missing translator.library",50));
    HALT(0)
  END;

CLOSE

  IF base#NIL THEN e.CloseLibrary(base) END;

END Translator.
```

Workbench:

```
MODULE Workbench; (* $Implementation- *)

IMPORT e: Exec,
       Dos,
     I: Intuition,
     s: SYSTEM;

CONST
  diskMagic * = 0E310H;
  diskVersion * = 1;
  gadgetBackFill * = {I.gadgHBox};
  noIconPosition * = MIN(LONGINT);

(* WBObjectType *)
  disk    * = 1;
  drawer  * = 2;
  tool    * = 3;
  project * = 4;
  garbage * = 5;
  device  * = 6;
  kick    * = 7;

(* MType *)
  pstd       * = 1;
  toolExit   * = 2;
  diskChange * = 3;
  timer      * = 4;
  closeDown  * = 5;
  ioProc     * = 6;

TYPE
  DiskObjectPtr * = POINTER TO DiskObject;
  DrawerDataPtr * = POINTER TO DrawerData;
  FreeListPtr * = POINTER TO FreeList;
  WBArgPtr * = POINTER TO WBArg;
  WBStartupPtr * = POINTER TO WBStartup;
  WBArg * = STRUCT
    lock * : Dos.FileLockPtr;
    name * : e.ADDRESS
  END;
  WBArgumentsPtr * = POINTER TO ARRAY 256 OF WBArg;
  WBStartup * = STRUCT
    message * : e.Message;
    process * : e.MsgPortPtr;
    segment * : e.BPTR;
```

```
    numArgs * : LONGINT;
    toolWindow * : e.ADDRESS;
    argList * : WBArgumentsPtr;
  END;
  FreeList * = STRUCT
    numFree * : INTEGER;
    memList * : e.List
  END;
  DiskObject * = STRUCT
    magic * : INTEGER;
    version * : INTEGER;
    gadget * : I.Gadget;
    type * : SHORTINT; (* WBObjectType *)
    defaultTool * : e.ADDRESS;
    toolTypes * : e.ADDRESS;
    currentX * : LONGINT;
    currentY * : LONGINT;
    drawerData * : DrawerDataPtr;
    toolWindow * : e.ADDRESS;
    stackSize * : LONGINT
  END;
  DrawerData * = STRUCT
newWindow * : I.NewWindow;
currentX * : LONGINT;
currentY * : LONGINT;
END;

CONST
drawerDataFileSize * = s.SIZE(DrawerData);

END Workbench.
```

15. Das Make-Utility OMake

Allgemeines:

Ein Make-Utility ist ein Programm, das das Neucompilieren von Modulen bei großen Programmprojekten stark vereinfacht. Wurden Module eines Projektes verändert, kann das Make dazu verwendet werden, alle Module, die bedingt durch diese Änderungen auch neu compiliert werden müssen, zu finden und automatisch neu zu compilieren.

Aufruf von OMake:

Im OEd kann OMake einfach durch Anwählen des Menüpunktes 'Make' im Menü 'Oberon' gestartet werden. Dabei muß sich der Quelltext des Hauptmoduls des Projektes im aktiven OEd-Fenster befinden. Wurde im Text etwas geändert, muß er zunächst gespeichert werden.

Start von der Workbench:

Von der Workbench kann OMake durch erweitertes Anwählen mit dem Quelltext des Hauptmoduls gestartet werden. Die Compileroptionen können über sogenannte Tool Types angegeben werden. Diese können, nachdem das Icon von OMake angewählt wurde, mit Info im Workbench-Menü geändert werden.

Start vom CLI:

Beim Start von OMake aus einem CLI heraus können noch eine Reihe von Optionen angegeben werden:

Aufruf:

OMake [c-svbcrnotpmd1238ig] [l-bsmdi] <source> [ALL] [DONTLINK] {OBJ <file>}

Dabei bedeutet:

c-[svbcrnotpmd1238ig]	beim Compilieren die angegebenen Optionen [svbcrnotpmd1238ig] verwenden
l-[bsmdi]	beim Linken die Optionen [bsmdi] verwenden
<source>	Hierhin sollte der Name des Hauptmoduls geschrieben werden
ALL	Diese Option erzwingt, daß alle Module neu compiliert werden, egal, ob dies nötig ist oder nicht
DONTLINK	Verhindert, daß OLink gestartet wird
OBJ	Dieser Parameter hat die gleiche Funktion wie der gleichnamige Parameter bei OLink. Dahinter kann eine weitere Objektdatei, z.B. die eines Assemblermoduls, angegeben werden, die zum Programm dazugelinkt werden soll.

Arbeitsweise von OMake:

OMake untersucht den Quelltext des Hauptmoduls und aller von diesem Modul direkt oder indirekt importierten Module. Die Quelltexte werden dabei in allen Pfaden, die in der Datei 'Oberon:Path' enthalten sind, gesucht.

Alle Module, deren Quelltexte jünger sind als ihre Objektdateien, werden neu compiliert. Gleiches geschieht mit Modulen, die Module importieren, deren Symboldateien jünger sind als die eigene Objektdatei.

Beispiele:

"OMake Test"

Compiliert alle Module, die Test.mod direkt oder indirekt imporiert und die seit der letzen Compilation verändert wurden. Danach wird Test evtl. neu gelinkt.

OMake c-dm l-dm Test ALL

Compiliert alle Module, die Test.mod importiert, und Test.mod selbst mit dem kleinen Daten- und Code-Modell neu und linkt Test mit diesen Speichermodellen.

Probleme:

Damit OMake korrekt funktionieren kann, dürfen die Erstellungsdaten der Objekt- und Symboldateien nicht verändert werden. Dies würde z.B. beim Kopieren der Dateien geschehen, wenn man die Option 'clone' beim COPY-Befehl nicht angibt.

Implementationslose Module (wie z.B. Exec.mod), die jünger sind als ihre Symboldateien, beim Compilieren jedoch keine neue Symboldatei erzeugen, werden immer neu compiliert.

16. Der Library-Linker LibLink

Mit diesem Programm ist es möglich, mit dem Amiga Oberon Compiler geschriebene Module zu Amiga-Libraries und -Devices zu Linken.

Installation:

Um LibLink verwenden zu können, müssen sich die Objektdateien LibraryHead.obj/.objs und LibOberonLib.obj/.objs in einem in Oberon:Path angegeben Verzeichnis befinden.

Start von LibLink:

LibLink kann nur vom CLI aus verwendet werden.

Aufruf:

```
LibLink {[[MAIN] <file>] [PROC <Module.Name>] [ICONS]
    [OPEN <Module.Name>] [CLOSE <Module.Name>]
    [VERSION <n>] [REVISION <n>] [SIZE <n>] [DEVICE]
    [TO <name.library>] [XREF <file>] [OBJ <file>]
    [SMALLDATA] [SMALL] [WITH <file>] }
```

Dabei bedeutet:

[MAIN] <file>

Das Hauptmodul (ohne Endung). Dieses Modul muß direkt oder indirekt alle anderen an der Library beteiligten Module importieren.

PROC <Module.Name>

Mit PROC können Prozeduren angegeben werden, die in die Library übernommen werden sollen. Die Prozeduren müssen qualifiziert angegeben werden, daß heißt Modulename, ein Punkt und der

Prozedurname. Die Prozedur muß im Oberon-Quelltext als exportiert gekennzeichnet sein. Es können auch Prozeduren aus verschiedenen Modulen angegeben werden. Dabei müssen jedoch alle beteiligten Module von dem bei MAIN angegebenen Modul importiert werden.

Es können beliebig viele Prozeduren angegeben werden. Die zuerst angegebenen Prozeduren bekommen die kleineren (absolut gesehen) Offsets zur Librarybasisadresse, d.h. die erste Prozedur -30, die zweite -36 etc.

Wer ein Device linken möchte, muß als erstes die zwei Prozeduren für BeginIO und AbortIO des Devices angeben.

ICONS

Wenn diese Option gesetzt ist, werden Icons erzeugt.

OPEN <Module.Name>

Mit OPEN kann optional eine parameterlose Prozedur mit einem Ergebnis vom Typ BOOLEAN angegeben werden, die bei jedem Öffnen der Library oder des Devices aufgerufen wird. Das Ergebnis der Prozedur ist FALSE, wenn die Library wegen Speichermangels oder ähnlichem nicht geöffnet werden konnte.

CLOSE <Module.Name>

Optionale parameterlose Prozedur, die bei jedem Schließen der Library aufgerufen wird.

VERSION <n>

Library-Versionsnummer. Wird VERSION nicht angegeben, bekommt die Library die Version 0.

REVISION <n>

Library-Revisionsnummer. Wird REVISION nicht angegeben, bekommt die Library die Revision 0.

SIZE <n>

Optionale Größe der Librarystruktur. Sie ist voreingestellt auf SYSTEM.SIZE(Exec.Library)=34. Möchte man noch weitere Daten darin speichern, muß man hier die Größe der verlängerten Librarystruktur angeben.

DEVICE

Wenn gesetzt, wird keine Library, sondern ein Device erzeugt.

TO <name.library>

Optionaler Name der zu erzeugenden Library bzw. des zu erzeugenden Devices. Wird er nicht angegeben, wird der Name von MAIN + '.library' verwendet.

XREF <file>

Name einer optionalen Cross-Reference-Datei. Dies ist eine ASCII-Datei die Informationen zur Library und die Offsets der Prozeduren enthält. Sie hilft beim Schreiben eines Interfacemoduls zum Aufruf der Libraryfunktionen.

OBJ <file>

Dieser Parameter entspricht dem OBJ von OLink und ermöglicht es, weitere Objektdateien, etwa von Assemblerroutinen, anzugeben, die zum Linken nötig sind.

SMALL

SMALL bewirkt, daß alle Datenhunks zu einem Hunk zusammengebunden werden, so daß die Library etwas kürzer wird.

SMALLDATA

Diese Option muß angegeben werden, wenn die Module der Library mit der Option '-d', also mit dem kleinen Datenmodell, compiliert wurden.

Bei einer solchen Library muß man besonders vorsichtig sein, da man selbst den Zeiger auf die globalen Variablen in jeder Libraryprozedur laden muß, bevor man auf globale Variablen zugreift. A5 wird von LibraryHead automatisch an die Variable mit dem Label 'LibraryHead.Globals' kopiert. Man sollte sich diese Variable also mit

```
VAR Globals["LibraryHead.Globals"]: LONGINT;
```

deklarieren und in jeder Prozedur mit 'SYSTEM.SETREG(13,Globals);' den Zeiger auf die Variablen nach A5 kopieren.

WITH <file>

WITH gibt eine Datei an, die weitere Parameter enthält. Die Parameter in der WITH-Datei entsprechen denen beim direkten Aufruf von LibLink. Da es bei größeren Libraries sehr viele Parameter werden können, sollte man sie in eine Datei schreiben. Die WITH-Datei darf aus beliebig vielen Zeilen bestehen und kann sogar mit WITH noch anderere WITH-Dateien aufrufen.

Die Reihenfolge der Parameter ist egal. Die Parameter in der CLI-Zeile sind mit denen in der WITH-Datei gleichgestellt. Der Parameter MAIN darf nicht weggelassen werden. Alle Parameter, mit der Aus-

nahme von PROC und WITH, sollten oder dürfen nicht mehrmals angegeben werden.

Wichtige Hinweise:

Die Module und alle von Ihnen importierten Module, die zu Libraries gebunden werden sollen, müssen folgende Eigenschaften haben:

- alle Prozeduren müssen reentrant sein. Das heißt, daß globale Variablen nur bedacht und mit äußerster Vorsicht verwendet werden dürfen.

- folgende Standardmodule dürfen nicht verwendet werden:

Break	BreakRq	FileSystem
io	NoGuru	NoGuruRq
OberonLib	RealInOut	SecureDos

- Da alle Module OberonLib automatisch importieren, wird diese beim Linken durch LibOberonLib ersetzt.

- Die Module sollten ohne Stackkontrolle compiliert werden. Es ist jedoch auch nicht schlimm, wenn die Stackkontrolle nicht abgeschaltet wurde. Dadurch wird lediglich Speicher und Zeit verschwendet.

- Die Module dürfen nur während der Ausführung der BEGIN-Teile mit HALT() abbrechen. überall sonst führt ein HALT zu Gurus oder ähnlichem.

- Laufzeitfehler führen zum Guru. Dies sollte jedoch kein Grund sein, den Überprüfungscode abzustellen, sondern vielmehr sollten die Module mit mehr Sorgfalt geschrieben und gut ausgetestet werden. Ein Fehler, der zu einem Guru führt ist immer besser als einer, den man nicht gleich bemerkt.

- Die Prozeduren, die als Libraryprozeduren dienen sollen, müssen die Register retten (am besten mit $SaveRegs+).

Libraries können auf ein paar Variablen zugreifen, die in den Assemblermodulen LibraryHead.obj und LibraryHead.objs definiert wurden:

```
VAR
  LibBase["LibraryHead.LibBase"]: Exec.LibraryPtr;
```

Zeigt auf die Library-Struktur dieser Library. Dieser Zeiger wird bei jedem Aufruf von Funktionen der Library in A6 übergeben.

```
SegList["LibraryHead.SegList"]: Exec.BPTR;
```

Ein Zeiger auf die Segmentliste dieser Library.

```
Globals["LibraryHead.Globals"]: LONGINT;
```

Globals existiert nur, wenn mit der Option '-d' compiliert und mit SMALLDATA gelinkt wurde. Es enthält die Basisadresse der globalen Variablen. Globals muß von jeder Prozedur vor der Verwendung globaler Variablen nach A5 kopiert werden.

Anhang A: Amiga Oberon 2.0

Diese überarbeitete Version des Amiga Oberon Compiler bietet eine Reihe an neuen Möglichkeiten, die das Arbeiten mit dem System erleichtern und die Programmierung des Amiga in Oberon vereinfachen. Spezielle Verbesserungen ermöglichen die Programmierung des AmigaOS 2.0 und die Ausnützung schnellerer Prozessoren, wie sie z.B. im Amiga 3000 vorhanden sind. Andere Erweiterungen verbessern den erzeugten Code ganz allgemein.

Änderungen am Compiler:

Start vom CLI:

Beim Start von CLI können nun folgende Parameter angegeben werden:

```
OBERON {-[svbcrnotpmd1238ig] |
        ['SET'|'CLEAR'] <Option> |
        <source>}
```

Mit den Optionen '-1', '-2' und '-3' kann der Compiler veranlaßt werden, Code zu erzeugen, der speziell für die Mikroprozessoren 68010, 68020 bzw. 68030 optimiert wird. Das Compilat ist danach nur auf Amigas lauffähig, die den entsprechenden Prozessor installiert haben, da Befehle, die nur auf diesen Prozessoren laufen, ausgenutzt werden, um schnellere Programme zu erzeugen.

Die Option '-8' bewirkt, daß Code für die Floating Point Unit (FPU) 68881 oder 68882 erzeugt wird. Dies sind Coprozessoren, die das Rechnen mit LONGREAL-Zahlen stark beschleunigen. Die so compilierten Module sind nur auf Rechnern mit FPU lauffähig.

Der Compiler benutzt bei kleinen Prozeduren automatisch Registerpa-

rameter und Registervariablen, damit diese Prozeduren schneller abgearbeitet werden können. Möchte man diese Optimierung unterdrücken, kann man die Option '-p' setzen.

Da das Dereferenzieren eines Zeigers, der einen ungeraden Wert enthält, auf den Prozessoren 68020 und 68030 nicht mehr automatisch zu einem Adressfehler führt, kann man durch Setzen der Option '-o' speziellen Code erzeugen, der ähnlich wie der Nil-Überprüfungscode prüft, ob Zeiger ungerade Werte enthalten und mit einem Laufzeitfehler abbricht, wenn dies der Fall ist.

Die Option '-g' erzeugt speziellen Code für den Run-Time-Source-Level-Debugger ODebug. Der Debugger kann als Zusatzpaket zum Compiler bei der A+L AG bestellt werden. Eine genauere Beschreibung dieser Option ist in der Dokumentation zu ODebug enthalten.

Mit "SET <Option>" und "CLEAR <Option>" kann eine Option für die bedingte Compilation gesetzt bzw. gelöscht werden. Genaueres zur bedingten Compilation steht weiter unten in diesem Text.

Start von der Workbench:

Beim Start des Compilers von der Workbench können die verschiedenen Optionen nun über sogenannte Tool Types gesetzt bzw. gelöscht werden. Die Tool Types können mit Info im Workbench-Menü geändert werden. Dabei stehen folgende Tool Types zu Verfügung:

STACKCHK	-s
OVFLCHK	-v
RANGECHK	-b
CASECHK	-c
RETURNCHK	-r
NILCHK	-n
ODDCHK	-o
TYPECHK	-t
AUTOREGPARS	-p

SMALLCODE	-m
SMALLDATA	-d
MC68010	-1
MC68020	-2
MC68030	-3
MC68881	-8
ICONS	-i
DEBUG	-g

Die Optionen entsprechen direkt den Optionen, die auch beim Start vom CLI angegeben werden können. Die CLI-Optionen sind daher oben jeweils hinter ihrer Tool Type Entsprechung aufgelistet.

Die bedingte Compilation:

Amiga Oberon 2.0 erlaubt die sogenannte bedingte Compilation. Dabei können bestimmte Teile des Quelltextes so markiert werden, daß sie nur beim Setzen bestimmter Optionen beim Compilieren beachtet werden. Dazu gibt es folgende neue Compileroptionen:

$IF <Option>
$IFNOT <Option>
$ELSE
$END

Dabei steht <Option> für eine beliebige Zeichenkette die aus Buchstaben und Ziffern besteht. Diese Optionen können beim Start vom Compiler mit "SET <Option>" gesetzt und mit "CLEAR <Option>" gelöscht werden. Wird eine Option beim Start nicht angegeben, ist sie automatisch gelöscht.

Kommt in einem Oberon-Quelltext in einem Kommentar "(* $IF <Option> *)" vor, werden die folgenden Anweisungen nur übersetzt, wenn <Option> beim Compilerstart gesetzt wurde. Durch "(* $END *)" wird diese bedingte Codeerzeugung wieder aufgehoben. "(* $ELSE *)" kehrt die letzte $IF-Anweisung um. "(* IFNOT <Option>

)" bewirkt das Gegenteil von "($IF <Option> *)", nämlich daß die folgendenden Anweisungen beachtet werden, wenn <Option> nicht gesetzt ist.

Beispiel:

```
MODULE  Test;

IMPORT io;

VAR string: ARRAY 80 OF CHAR;

CONST

(* $IF English *)

  hallo = "Hello!";
  nochmal = "again?";

(* $ELSE *)

  hallo = "Hallihallo";
  nochmal = "nochmal?";

(* $END *)

BEGIN
  REPEAT
    io.WriteString(hallo); io.WriteLn;
    io.WriteString(nochmal);  io.ReadString(string);
(* $IFNOT English *)
  UNTIL (string#"ja") AND (string#"j");
(* $ELSE *)
  UNTIL (string#"yes") AND (string#"y");
(* $END *)
END Test.
```

Aus diesem kleinen Programm kann nun durch Neucompilation eine englische und eine deutsch Version erzeugt werden, ohne daß der Quelltext verändert wird:

Beim Start des Compilers mit

Oberon SET English Test.mod

wird eine englischsprachige Version erzeugt, während bei

Oberon CLEAR English Test.mod

oder einfach

Oberon Test.mod

ein deutschsprachiges Programm erzeugt wird. Auf diese Weise können auch verschiedene Versionen eines Programms, wie eine eingeschränkte Demo-Version und eine voll funktionsfähige Version, aus dem gleichen Quelltext nur durch Setzen bzw. Löschen von Optionen erzeugt werden.

Das Modul MATHLIB:

Wird in einem Programm viel mit LONGREAL-Zahlen gerechnet, kann die Geschwindigkeit durch Ausnutzung einer FPU stark verbessert werden. Dies geschieht durch Setzen der Compileroption '-8' bzw. des Tool Types 'MC68881=TRUE'. Dabei werden jedoch lediglich die Grundrechenarten und die Funktionen ABS() und ENTIER() in Befehle der FPU übersetzt. Die Prozessoren MC68881 bzw. MC68882 besitzen jedoch viele weitere Befehle, wie z.B. die trigonometrischen Funktionen.

Damit diese Befehle auch voll ausgenutzt werden können, existiert bei gesetzter Option '-8' ein compilerinternes Modul (ähnlich wie das Modul SYSTEM) mit dem Namen MATHLIB, das genau diese Funktionen enthält. Eine von ModToDef erzeugte Definition dieses Moduls würde folgendermaßen aussehen:

```
DEFINITION MATHLIB;

PROCEDURE ACOS    (x: LONGREAL): LONGREAL;
PROCEDURE ASIN    (x: LONGREAL): LONGREAL;
PROCEDURE ATAN    (x: LONGREAL): LONGREAL;
PROCEDURE ATANH   (x: LONGREAL): LONGREAL;
PROCEDURE COS     (x: LONGREAL): LONGREAL;
PROCEDURE COSH    (x: LONGREAL): LONGREAL;
PROCEDURE ETOX    (x: LONGREAL): LONGREAL;
PROCEDURE LOG10   (x: LONGREAL): LONGREAL;
PROCEDURE LOG2    (x: LONGREAL): LONGREAL;
PROCEDURE LOGN    (x: LONGREAL): LONGREAL;
PROCEDURE SIN     (x: LONGREAL): LONGREAL;
PROCEDURE SINH    (x: LONGREAL): LONGREAL;
PROCEDURE SQR     (x: LONGREAL): LONGREAL;
PROCEDURE SQRT    (x: LONGREAL): LONGREAL;
PROCEDURE TAN     (x: LONGREAL): LONGREAL;
PROCEDURE TANH    (x: LONGREAL): LONGREAL;
PROCEDURE TENTOX  (x: LONGREAL): LONGREAL;
PROCEDURE TWOTOX  (x: LONGREAL): LONGREAL;
PROCEDURE INT     (x: LONGREAL): LONGREAL;

END MATHLIB.
```

Die Funktionen berechnen dabei folgendes:

SIN, COS, TAN	sin(x), cos(x), tan(x)
ASIN, ACOS, ATAN	arcsin(x), arccos(x), arctan(x)
SINH, COSH, TANH	sinh(x), cosh(x), tanh(x)
ATANH	artanh(x) (Hyperbelfunktionen)
TENTOX	10^x
TWOTOX	2^x
ETOX	e^x
LOG10	$\log_{10}(x)$
LOG2	$\log_2(x)$
LOGN	ln(x)
SQR	x^2
SQRT	$x^{0.5}$
INT	ENTIER(x), für x>=0; -ENTIER(-x), für x<0

Damit Programme, die das Modul MATHLIB benutzen, auch so com-

piliert werden können, daß sie ohne FPU lauffähig sind, wurde die Modulbibliothek um das gleichnamige Modul MATHLIB erweitert, das jedoch nicht compilerintern ist. Wird ein Programm ohne Ausnutzung der FPU compiliert, wird dieses Modul automatisch importiert. Dieses MATHLIB-Modul ruft vor allem die Prozeduren des Moduls MathIEEEDoubTrans auf, es bringt also keinen Geschwindigkeitsvorteil. Dennoch wird empfohlen, MATHLIB zu verwenden, da nur so die volle Ausnutzung einer FPU möglich wird.

Die Operatoren MOD und DIV:

Amiga Oberon 1.17.1 hielt sich bei der Berechnung der Werte, die die Funktionen MOD und DIV bei negativen Operanden als Ergebnis liefern, an den Oberon-Report und nicht an den Revised Oberon-Report. Dies ist nun geändert. Es gilt:

a = (a DIV b) * b + (a MOD b)
0 <= (a MOD b) < b oder 0 >= (a MOD b) > b

Erweiterbare STRUCTs:

Mit dieser Compilerversion können STRUCTs, ähnlich wie dies bei RECORDs möglich ist, erweitert werden. Dazu kann bei der Definition eines STRUCTs das erste Element, das auch ein STRUCT sein muß, in Klammern geschrieben werden. Durch diese Schreibweise wird die Hierarchie deutlich: Die neue Struktur erbt die Elemente und Eigenschaften der alten Struktur. Auf die Elemente muß jedoch, anders als bei RECORDs, über den Namen des ersten Elements zugegriffen werden.

STRUCTs sind zuweisungskompatibel zu ihren Basistypen. Zeiger auf STRUCTs können sowohl an ihre Basistypen als auch an ihre Erweiterungen zugewiesen werden. Außerdem kann jetzt auch bei Zeigern auf STRUCTs der Typ mit Hilfe des Type-Guards oder mit der WITH-Anweisung auf den Typ einer Erweiterung gesetzt werden. Allerdings

kann der Compiler bei STRUCTs keinen Code erzeugen, der die Korrektheit des Typwechsels prüft. Entsprechend trägt auch bei Zuweisungen von Zeigern auf STRUCTs an Zeiger auf deren Erweiterungen der Programmierer die Verantwortung für die Korrektheit dieser Zuweisung.

Beispiel:

Im Amiga-Interface-Modul Exec ist eine Message und ein MsgPort nun eine Erweiterung einer Node. Diese Typen werden daher folgendermaßen definiert:

```
TYPE
  MsgPort * = STRUCT (node * : Node)
    flags * : SHORTINT;
    sigBit * : SHORTINT;
    sigTask * : TaskPtr;
    msgList * : List;
  END;
  Message * = STRUCT (node * : Node)
    replyPort * : MsgPortPtr;
    length * : INTEGER;
  END;
```

In einem Programm kann man nun z.B. die Länge der ersten an einem Port wartenden Message folgendermaßen abfragen:

```
VAR
  port: Exec.MsgPortPtr;
  len: INTEGER;

BEGIN
  IF ~ ExecSupport.ListEmpty(port.msgList) THEN
    len := port.msgList.first.head(Message).length;
  END;
END;
```

Enstsprechend sind nun viele Strukturen in den Amiga-Interface-Modulen Erweiterungen anderer Strukturen, so daß man meist ohne die Funktion sys.VAL() auskommt. Dennoch sollte man sich immer vergewissern, ob ein STRUCT wirklich erweitert und ein Zugriff auf die Elemente der Erweiterung legal ist.

Listenparameter:

Das AmigaOS 2.0 kennt eine Reihe von Prozeduren, die beliebig lange Listen von Parametern erwarten. Damit mit diesen Prozeduren einfach und sauber auch in Oberon gearbeitet werden kann, mußte die Syntax bei der Deklaration und beim Aufruf solcher Prozeduren leicht verändert werden.

Beispiel:

Die Prozedur AllocAslRequestTags der asl.library des AmigaOS 2.0 bekommt eine beliebig lange sogenannte Tag-Liste als letzten Parameter. Bei der Deklaration dieser Prozedur im Modul ASL wird sie daher folgendermaßen deklariert:

```
PROCEDURE AllocAslRequestTags*{asl,-48}(
                    type{0}:  LONGINT;
                    tag1{8}.. : Utility.Tag): e.APTR;
```

Der Listenparameter wird mit 2 Punkten hinter dem Parameternamen und hinter dem Register markiert. Es muß der letzte Parameter und ein Registerparameter sein. Normale Oberon-Prozeduren können keine Listenparameter verarbeiten, da dies von der Sprache nicht vorgesehen ist.

AllocAslRequestTags kann nun mit beliebig vielen Tags als Parameter aufgerufen werden:

```
fileReq = asl.AllocAslRequestTags(asl.fileRequest,
                    asl.leftEdge,   20,
                    asl.topEdge,    20,
                    asl.width,      300,
                    asl.height,     200,
                    Utility.done);
```

Einschränkungen:

Die Liste der Listenparameter darf maximal 512 Bytes Speicher belegen. Dies sollte für die meisten Anwendungen genügen.

Innerhalb der Listenparameter dürfen keine Prozeduraufrufe, außer Aufrufe von Standardprozeduren, vorkommen. Soll das Ergebnis einer Prozedur als einer der Parameter übergeben werden, muß die Prozedur daher vorher aufgerufen und ihr Ergebnis in einer Variablen zwischengespeichert werden.

Registerparameter:

Der Amiga Oberon Compiler erzeugt mit dieser Version automatisch Registerparameter, wo dies sinnvoll und nötig ist. Registerparameter kann man jedoch auch durch Angabe der Registernummer hinter dem Parameternamen in geschweiften Klammern erzwingen. Die Register D0 bis D7 haben dabei die Nummern 0 bis 7, A0 bis A7 die Nummern 8 bis 15. Wenn Code für die FPU erzeugt wird, können auch die FPU-Register FP0 bis FP7 mit den Nummern 16 bis 23 verwendet werden.

Mit dieser Compilerversion sind so erzwungene Registerparameter, mit der Ausnahme D0 und D1, vom Überschreiben geschützt. Es muß also keine Kopie der Registervariablen mehr erzeugt werden. Dies ist vor allem beim Schreiben von Amiga-Libraries wichtig. Lediglich D0 und D1 müssen weiterhin vorsichtig gehandhabt werden.

Da der Compiler die Register A5 bis A7 für besondere Zwecke vorgesehen hat, dürfen diese auch nicht zur Übergabe von Registerparametern verwendet werden.

Die Prozeduren REG() und SETREG() aus dem compilerinternen Modul SYSTEM akzeptieren nun auch die Registernummern 16 bis 23, wenn mit Option '-8' compiliert wird.

Compileroptionen:

Es gibt eine neue stapelbare Compileroption "$AutoRegPars". Die Voreinstellung ist '+'. Mit dieser Option kann für einzelne Prozeduren oder Teile eines Programms die automatische Verwendung von Registerparametern und Registervariablen abgeschaltet werden. Dies wird z.B. nötig, wenn man die Adresse einer Variablen mit SYSTEM.ADR() bestimmen möchte, was bei Registervariablen nicht möglich ist und zu der Fehlermeldung 'Adresse nicht ladbar' führt. Bei manchen Prozeduren, die viele andere Prozeduren aufrufen, kann es auch sinnvoll sein, keine Registervariablen zu verwenden, da der Aufwand beim Speichern und Wiederholen der Variablenwerte vor und nach Prozeduraufrufen zu groß ist.

Die neue stapelbare Compileroption $Debug ist nur im Zusammenhang mit dem Debugger ODebug wichtig und wird daher in der Anleitung zum Debugger beschrieben.

Symboldateien:

Durch die Änderungen an der Sprache hat sich das Format der Symboldateien geändert. Daher müssen alle mit Oberon 1.17.1 compilierten Module neu compiliert werden. Es ist dabei am besten, vorher alle alten Symboldateien zu löschen.

Änderungen am Editor:

Start vom CLI:

Beim Start von OEd vom CLI aus können nun die Compiler-Optionen vorbelegt werden. Die Aufrufsyntax wurde dazu folgendermaßen geändert:

```
OEd     {-{i|t#|x#|y#|w#|h#|d#|s|l}}
        [c-{s|v|b|c|r|n|t|m|d|1|2|3|8|g}]
        {<Text>}
```

Mit 'c-...' können die Optionen, die beim Start des Compilers verwendet werden sollen, vorbelegt werden. Die Bedeutung der einzelnen Buchstaben entspricht der beim Start des Compilers.

Start von der Workbench:

Die Compileroptionen können beim Start von der Workbench-Oberfläche aus nun mit Hilfe von Tool Types gesetzt werden. Die Namen der Tool Types entsprechen dabei denen der Tool Types des Compilers selbst.

Die OEd-Fenster:

Am rechten Rand der von OEd geöffneten Fenster befindet sich nun ein Scroll-Gadget, mit dem man sich schnell durch den Text bewegen kann und leicht die Position des im Augenblick angezeigten Textteils im Gesamttext sieht.

Mit den Pfeilen unter dem Scroll-Gadget kann man sich seitenweise durch den Text bewegen, wie es auch mit <Alt> und den Pfeiltasten möglich ist.

Find / Replace:

Die Funktionen zum Suchen- und Ersetzen im Text wurden stark überarbeitet. Wählt man *Find* im *Search*-Menü, wird ein Fenster mit mehreren Gadgets geöffnet. In das aktive Text-Gadget kann eine Zeichenkette, nach der gesucht werden soll, eingegeben werden.

Mit den drei Gadgets darunter kann gewählt werden, wie nach der Zeichenkette gesucht werden soll. Das erste Gadget wählt, ob zwischen Groß- und Kleinschreibung unterschieden werden soll ('Case-Sensitive') oder ob Groß- und Kleinbuchstaben als gleich betrachtet werden sollen ('nicht Case-Sens.'). Das nächste Gadget wählt die Richtung, in der der Text durchsucht werden soll, 'vorwärts' steht dabei für rechts und unterhalb des Cursors, 'rückwärts' für links und oberhalb des Cursors. Das letzte Gadget wählt, ob zeichenweise oder wortweise gesucht werden soll. Beim wortweisen Suchen werden Zeichenketten, die in einem längeren Wort enthalten sind (wie z.B. 'Buch' in 'Kleinbuchstaben' bei nicht Case-sensitivem Suchen) nicht gefunden. Beim wortweisen Suchen kann nur nach Buchstaben und Ziffern gesucht werden, da alle andere Zeichen als Separatoren zwischen Wörtern angesehen werden.

Mit den 2 Gadgets am unteren Rand des Fenster kann das Suchen gestartet ('ok'), oder der Vorgang abgebrochen ('Abbruch') werden.

Wählt man *FindRep* im *Search*-Menü, wird ein Fenster mit 2 Text-Gadgets geöffnet. In das erste Gadget muß der Text, nach dem gesucht werden soll, und in das zweite der Text, durch den der erste Text ersetzt werden soll, eingegeben werden.

Unter den Text-Gadgets befinden sich vier weitere Gadgets. Die ersten drei dieser Gadgets entsprechen denen bei *Find* und haben die gleiche Bedeutung. Das letzte Gadget wählt, ob nur einmal das nächste Vorkommen des Such-Textes ersetzt werden soll ('einzeln'), oder ob der gesamte Text durchgegangen werden soll ('alle').

Ist 'alle' gewählt, wird jede gefunde Zeichenkette einzeln angezeigt und dabei ein Fenster mit vier Gadgets geöffnet. Durch Anklicken eines der Gadgets kann man wählen, was geschehen soll: 'ja!' ersetzt den Text durch den vorhin eingegebenen Ersatz-Text. 'nein!' läßt den Text unverändert und sucht nach dem nächsten Vorkommen des Such-Textes. 'alle' ersetzt alle weiteren Vorkommen des Such-Textes, ohne weiter nachzufragen. 'Cancel' bricht den Vorgang ab. Statt eines der vier Gadgets anzuklicken, kann auch einfach der Anfangsbuchstabe der gewünchsten Funktion eingetippt werden, also 'j', 'n', 'a' oder 'c'.

Eigener Screen:

Soll OEd einen eigenen Screen öffnen und arbeitet man unter AmigaOS 2.0, wird ein sogenannter Public-Screen mit dem Namen 'Oberon Screen' geöffnet. Auf diesem Screen können noch weitere Programme ihre Fenster öffnen. So öffnet z.B. ein später gestarteter OEd seine Fenster auf diesem Screen, und öffnet keinen neuen Screen. Gleichermaßen wird dieser Screen von dem Debugger ODebug verwendet.

Options-Menü:

Das *Options*-Menü wurde um folgende Menüpunkte erweitert:

mc68010, mc68020 und mc68030:

Diese Optionen wählen die Codeoptimierung für die Prozessoren MC68010, MC68020 bzw. MC68030. Ist keiner der Prozessoren angewählt, wird Code für den MC68000 erzeugt. Dies Entspricht den Compileroptionen '-1', '-2' und '-3'.

mc68881:

Sollen die Befehle der FPU MC68881 bzw. MC68882 ausgenutzt

werden, muß diese Option angewählt sein. Dieser Menüpunkt entspricht der Compileroption '-8'.

Debug:

Diese Option entspricht '-g' beim Start des Compilers vom CLI. Sie wird erst im Zusammenhang mit dem Debugger wichtig.

Die Modulbibliothek:

FileSystem:

Dieses Modul kennt eine neue Funktion:

```
PROCEDURE OpenReadWrite(VAR file: File;
                        name: ARRAY OF CHAR): BOOLEAN;
```

Es wird die bereits existierende Datei mit dem Namen *name* zum gemischten schreibenden und lesenden Zugriff geöffnet. Existiert *name* noch nicht, wird eine neue Datei erzeugt.

Display:

Auch das Modul Display wurde um ein Prozedur ergänzt:

```
PROCEDURE OpenWindowX*(win: WindowPtr;
                       gadg: I.GadgetPtr;
                       gzz: BOOLEAN;
                       title: ARRAY OF CHAR;
                       x,y,w,h: INTEGER;
                       screen: I.ScreenPtr;
                       activate: BOOLEAN): BOOLEAN;
```

Diese Prozedur ermöglicht das Öffnen eines Fensters wie Dis-

play.OpenWindow(), es können jedoch mehr Parameter selbst definiert werden. *gadg* ist eine optionale Liste von Gadgets, die das Fenster bekommen soll.

gzz gibt an, ob ein Gimme-Zero-Zero-Fenster geöffnet werden soll. Dies ist bei allen mit Display.OpenWindow() geöffneten Fenstern der Fall. Diese Fenster haben den Vorteil, daß man auch über den Rand des Innenbereichs des Fensters hinaus zeichnen kann, ohne den Rahmen des Fensters zu zerstören. Das Gezeichnete wird 'geclippt', das heißt, am Rand abgeschnitten. Nachteil dieser Fenster ist, daß sie mehr Speicher benötigen und mehr Rechenzeit verbrauchen. Ist man sicher, daß man nur im inneren Bereich des Fenster zeichnet, kann man *gzz* auf FALSE setzen. Das Modul Display übernimmt aber auch dann die Umrechnung der Koordinaten, so daß der Punkt (0/0) in der linken oberen Ecke des inneren Bereichs des Fensters liegt.

Die Interface-Module:

Die Interface-Module für das Amiga-Betriebssystem wurden völlig überarbeitet. Sie entsprechen nun den C-Include-Dateien des AmigaOS 2.0. Wer diese Version des AmigaOS installiert hat, kann auf die hier neu hinzugekommenen Funktionen zugreifen.

Die Interface-Module für Amiga-Libraries, die es auch unter AmigaOS 1.2 und 1.3 bereits gab, öffnen die Libraries automatisch mit der unter 1.2 gültigen Versionsnummer (33). Die Prozeduren, die mit dem AmigaOS 2.0 hinzugekommen sind, sind in den Modulen durch einen Kommentar gekennzeichnet. Sie dürfen nur verwendet werden, nachdem man die Versionsnummer der Library geprüft hat. Die Version muß mindestens 36 sein.

Beispiel:

```
MODULE Test;

IMPORT Intuition;
```

```
VAR window: Intuition.WindowPtr;

BEGIN
  IF Intuition.libNode.version<36 THEN
    io.WriteString("Kauf Dir 'nen A3000!\n"); HALT(20);
  ELSE
    window := I.OpenWindowTags(nw);
    ...
  END;
CLOSE
  IF window#NIL THEN I.CloseWindow(window) END;
END Test.
```

Diese Versionsüberprüfung ist bei den Libraries ASL, Commodities, GadTools, IFFParse und Utility nicht nötig. Da diese Libraries erst ab AmigaOS 2.0 existieren, werden sie gleich mit Versionsnummer 36 geöffnet. Utility überprüft jedoch nicht, ob das Öffnen erfolgreich war, da Utility z.B. von Dos importiert wird und somit alle Programe, die Dos importieren, nicht mehr unter AmigaOS 1.3 lauffähig wären. Vor der Verwendung von Prozeduren aus Utility muß also geprüft werden, ob Utility.base#NIL ist.

Manche Prozeduren haben mit AmigaOS 2.0 Ergebnisse bekommen, die sie unter 1.3 nicht hatten. So gibt z.B. Intuition.CloseScreen() nun ein BOOLEAN zurück. Für die wichtigsten dieser Prozeduren existieren Synonyme ohne Ergebnis, die mit 'Old' vorne im Namen gekennzeichnet sind. So kann Intuition.OldCloseScreen() verwendet werden, wenn man das Ergebnis ignorieren möchte.

Andere Änderungen an den Modulen wurden vorgenommen, damit diese eher ihren C-Äquivalenten entsprechen. So gibt es das Modul Resources nicht mehr. Es wurde in die Module Disk, Misc und Potgo aufgespalten. Die Module Printer und PrtBase wurden in dem Modul Printer zusammengefaßt. Eine weitere Änderung, die evtl. zu Verwirrung führen kann, sind die Interrupt-Flags, die nicht mehr in Exec, sondern nun, wie bei den Original-Include-Dateien, in Hardware definiert werden.

Die *portName*- und *taskName*-Parameter der Prozeduren *CreatePort()* und *CreateTask()* aus ExecSupport haben nun ARRAY OF CHAR-Parameter, und verlangen nicht mehr die Adresse der Zeichenketten. Um einen namenlosen nicht öffentlichen Port zu erzeugen, ruft man ExecSupport.CreatePort mit einem leeren String als Portname auf.

Bei anderen kleineren Änderungen in den Modulen hilft meist ein kurzer Blick in die Quelltexte der Module. Die Module enthalten nun sehr viel mehr Kommentare, so daß man mit ihnen einfacher arbeiten kann.

Bezeichername:	Modul:	Seite:
abc	Hardware	14/59
AbleICR	Resources	14/119
abnc	Hardware	14/59
abort	Parallel	14/108
abort	Serial	14/122
aborted	Exec	14/24
abortIO	Exec	14/24
AbortIO	Exec	14/27
AbortIO	ExecSupport	14/32
Abs	MathFFP	14/101
Abs	MIDoubBas	14/102
absJoystick	GamePort	14/37
accessRead	Dos	14/9
accessWrite	Dos	14/9
ack	ASCII	13/3
Acos	MIDbTrans	14/104
Acos	MathTrans	14/105
actionEnd	Dos	14/13
actionNotKnown	Dos	14/11
activate	Intuition	14/80
ActivateGadget	Intuition	14/90
ActivateWindow	Intuition	14/90
active	Exec	14/24
active	Parallel	14/108
active	Resources	14/117
active	Serial	14/122
activewindow	InputEvent	14/71
activeWindow	Intuition	14/79
Add	AVL	13/4
Add	MathFFP	14/101
Add	MIDoubBas	14/102
AddAnimOb	Graphics	14/50
AddBefore	Lists	13/37
AddBehind	Lists	13/37
AddBob	Graphics	14/50
addChangeInt	TrackDisk	14/125
AddConfigDev	Expansion	14/35
AddDevice	Exec	14/27
AddDosNode	Expansion	14/35
added	Exec	14/22
AddFont	Graphics	14/50
AddFreeList	Icon	14/68
AddGadget	Intuition	14/90
AddGList	Intuition	14/90
addHandler	Input	14/70
AddHead	Lists	13/37
AddHead	Exec	14/27
AddICRVector	Resources	14/120
AddIntServer	Exec	14/27
additive	PrtBase	14/115
AddLibrary	Exec	14/27
AddMemList	Exec	14/27
AddPort	Exec	14/27
addRequest	Timer	14/124
addResetHandler	KeyBoad	14/96
AddResource	Exec	14/27
ADDRESS	Exec	14/19
AddSemaphore	Exec	14/27
AddTail	Lists	14/37
AddTail	Exec	14/27
AddTask	Exec	14/27
AddTime	Timer	14/124
AddVSprite	Graphics	14/50
adkSet	Hardware	14/58
Alert	Exec	14/28
alertLayersNoMem	Graphics	14/45
alloc0	Resources	14/117
alloc1	Resources	14/117
alloc2	Resources	14/117
alloc3	Resources	14/117
AllocAbs	Exec	14/28
allocate	Audio	14/2
Allocate	Exec	14/28
allocated	GamePort	14/37
AllocBoardMem	Expansion	14/35
AllocConfigDev	Expansion	14/35
AllocEntry	Exec	14/28
AllocExpansionMem	Expansion	14/35
allocFailed	Audio	14/2
allocMaxprec	Audio	14/2
AllocMem	Exec	14/28
allocMinprec	Audio	14/2
AllocMiscResource	Resources	14/120
AllocPotBits	Resources	14/120
AllocRaster	Graphics	14/50
AllocRemember	Intuition	14/91
AllocSignal	Exec	14/28
AllocTrap	Exec	14/28
AllocUnit	Resources	14/120
AllocWBObject	Icon	14/68
allowNon35	TrackDisk	14/125
AlohaWorkbench	Intuition	14/91
alphaP101	Intuition	14/83
alrm	Hardware	14/65

alt	KeyMap	14/97
altKeyMap	Intuition	14/75
altLeft	Intuition	14/79
altRight	Intuition	14/79
amiga	Resources	14/118
amigaKeys	Intuition	14/80
amigaLeft	Intuition	14/79
amigaRight	Intuition	14/80
anbc	Hardware	14/59
anbnc	Hardware	14/59
AndRectRegion	Graphics	14/50
AndRegionRegion	Graphics	14/50
anfracsize	Graphics	14/41
Animate	Graphics	14/50
AnimComp	Graphics	14/42
AnimCompPtr	Graphics	14/38
animhalf	Graphics	14/41
AnimOb	Graphics	14/43
AnimObPtr	Graphics	14/38
aORb	Hardware	14/59
aORc	Hardware	14/59
Append	Strings	13/57
AppendChar	Strings	13/57
archive	Dos	14/9
AreaDraw	Graphics	14/50
AreaEllipse	Graphics	14/50
AreaEnd	Graphics	14/50
AreaInfo	Graphics	14/45
AreaInfoPtr	Graphics	14/38
AreaMove	Graphics	14/50
areaOutline	Graphics	14/47
asciiDel	InputEvent	14/71
asciiFirst	InputEvent	14/71
asciiLast	InputEvent	14/71
ash1	Hardware	14/59
ash2	Hardware	14/59
ash4	Hardware	14/59
ash8	Hardware	14/59
Asin	MIDbTrans	14/104
Asin	MathTrans	14/105
askCType	GamePort	14/37
askDefaultKeyMap	Console	14/5
AskFont	Graphics	14/50
askKeyMap	Console	14/5
AskSoftStyle	Graphics	14/50
askTrigger	GamePort	14/37
aspect	Printer	14/112
aspectHoriz	Intuition	14/85
aspectVert	Intuition	14/85
Assert	NoGuru	13/43
Assert	Requests	13/54
Atan	MIDbTrans	14/104
Atan	MathTrans	14/105
aTOd	Hardware	14/59
att	Hardware	14/62
AttemptLockLayer-Rom	Graphics	14/50
AttemptSemaphore	Exec	14/28
aud0	Hardware	14/61
aud0i	Exec	14/19
aud1	Hardware	14/61
aud1i	Exec	14/19
aud2	Hardware	14/61
aud2i	Exec	14/19
aud3	Hardware	14/61
aud3i	Exec	14/19
AudioChannels	Hardware	14/58
AudioInfo	Hardware	14/58
aul	Hardware	14/59
autoBackPen	Intuition	14/77
autoDrawMode	Intuition	14/77
autoFrontPen	Intuition	14/77
autoinit	Exec	14/25
autoITextFont	Intuition	14/78
autoKnob	Intuition	14/77
autoLeftEdge	Intuition	14/78
autoNextText	Intuition	14/78
AutoRequest	Intuition	14/91
autoTopEdge	Intuition	14/78
AvailFont	DiskFont	14/7
AvailFontHeader	DiskFont	14/7
AvailFontHeaderPtr	DiskFont	14/8
AvailFonts	DiskFont	14/8
AvailMem	Exec	14/28
aXORc	Hardware	14/59
b2Bobber	Graphics	14/41
b2Norm	Graphics	14/41
b2Swap	Graphics	14/41
backDrop	Intuition	14/80
BackPen	Display	13/15
backSaved	Graphics	14/41
Backward	FileSystem	13/28
badDimension	Printer	14/112
badDriveType	TrackDisk	14/126
badHdrSum	TrackDisk	14/125
badLength	Exec	14/24

badSecHdr	TrackDisk	14/126
badSecId	TrackDisk	14/125
badSecPreamble	TrackDisk	14/125
badSecSum	TrackDisk	14/125
BadStatus	SCSI	14/121
badStreamName	Dos	14/11
badUnitNum	TrackDisk	14/126
base	DiskFont	14/8
base	Exec	14/23
base	Expansion	14/35
base	Icon	14/68
base	Layers	14/99
base	MathFFP	14/101
base	MIDoubBas	14/102
base	MIDbTrans	14/104
base	MathTrans	14/105
base	Translator	14/127
baud110	Intuition	14/85
baud1200	Intuition	14/85
baud19200	Intuition	14/85
baud2400	Intuition	14/85
baud300	Intuition	14/85
baud4800	Intuition	14/85
baud9600	Intuition	14/85
baudMidi	Intuition	14/85
baudMismatch	Serial	14/123
bDrawn	Graphics	14/42
Beep	Beep	13/10
beeping	Intuition	14/81
beginIO	Exec	14/24
BeginIO	ExecSupport	14/32
beginning	Dos	14/9
BeginRefresh	Intuition	14/91
BeginUpdate	Layers	14/99
BehindLayer	Layers	14/99
bel	ASCII	13/3
bgr	PrtBase	14/115
bgrw	PrtBase	14/115
bgrWb	PrtBase	14/115
bigdots	Display	13/15
bindTime	Expansion	14/34
bitClr	Hardware	14/62
BitMap	Graphics	14/43
BitMapPtr	Graphics	14/38
bitSet	Hardware	14/62
BitTable	KeyMap	14/97
BitTablePtr	KeyMap	14/97
black	Console	14/5
blackAndWhite	PrtBase	14/115
blackBg	Console	14/5
blit	Exec	14/19
blithog	Hardware	14/61
blitMsgFault	Graphics	14/44
blitReverse	Hardware	14/59
blitter	Hardware	14/61
BltBitMap	Graphics	14/50
BltBitMapRastPort	Graphics	14/51
BltClear	Graphics	14/51
bltdone	Hardware	14/61
BltMaskBitMap-RastPort	Graphics	14/51
Bltnode	Hardware	14/60
BltnodePtr	Hardware	14/60
bltnzero	Hardware	14/61
BltPattern	Graphics	14/51
BltTemplate	Graphics	14/51
blue	Console	14/5
blueBg	Console	14/5
bms	Printer	14/111
BndryOff	Graphics	14/56
Bob	Graphics	14/42
bobIsComp	Graphics	14/42
bobNix	Graphics	14/42
BobPtr	Graphics	14/38
bobsAway	Graphics	14/42
bobUpdate	Graphics	14/41
bold	Console	14/5
bold	Graphics	14/46
BoldOff	Display	13/15
BoldOn	Display	13/15
boolExtend	Intuition	14/75
boolGadget	Intuition	14/76
BoolInfo	Intuition	14/76
boolMask	Intuition	14/76
BoolToLong	Intuition	13/37
BootBlock	BootBlock	14/3
bootNode	Exec	14/19
bootSects	BootBlock	14/3
bootTime	Expansion	14/34
Border	Intuition	14/78
borderHit	Graphics	14/39
borderless	Intuition	14/80
BorderPtr	Intuition	14/73
bottomBorder	Intuition	14/75
bottomHit	Graphics	14/39
Box	Display	13/15

BPTR	Exec	14/19
bpu0	Hardware	14/60
bpu1	Hardware	14/60
bpu2	Hardware	14/60
break	Serial	14/122
BreakPoint	Requests	13/54
broken	Display	13/15
brother15XL	Intuition	14/83
bs	ASCII	13/3
bsh1	Hardware	14/59
bsh2	Hardware	14/59
bsh4	Hardware	14/59
bsh8	Hardware	14/59
BSTR	Dos	14/9
buf1024	Intuition	14/85
buf16000	Intuition	14/85
buf2048	Intuition	14/85
buf4096	Intuition	14/85
buf512	Intuition	14/85
buf8000	Intuition	14/85
bufErr	Serial	14/123
bufferMemory	Printer	14/112
bufOverflow	Serial	14/123
bufrRead	Serial	14/122
bufSize	PrtBase	14/114
bufSizeBits	Intuition	14/85
bufTooBig	Parallel	14/108
BuildSysRequest	Intuition	14/91
BumpRevision	Icon	14/68
busWidth	Expansion	14/34
busy	Serial	14/122
bw	PrtBase	14/115
bWaiting	Graphics	14/42
bwAlpha	PrtBase	14/115
bwGfx	PrtBase	14/115
byteWide	Expansion	14/34
c0First	InputEvent	14/71
c0Last	InputEvent	14/71
c1First	InputEvent	14/71
c1Last	InputEvent	14/71
cam	Printer	14/111
can	ASCII	13/3
cancel	Printer	14/112
cantAlloc	Narrator	14/106
cantlock	FileSystem	13/28
cantopen	FileSystem	13/28
capsLock	InputEvent	14/72
carrierDetect	Serial	14/122
Cause	Exec	14/28
cbmMps1000	Intuition	14/83
CBump	Graphics	14/51
CDInputHandler	Console	14/6
Ceil	MathFFP	14/101
Ceil	MIDoubBas	14/102
CEND	Graphics	14/56
center	Printer	14/112
chainedConfig	Expansion	14/34
changed	Exec	14/23
changeNum	TrackDisk	14/125
ChangeSprite	Graphics	14/51
changeState	TrackDisk	14/125
channelStolen	Audio	14/2
checked	Intuition	14/74
CheckIO	Exec	14/28
checkIt	Intuition	14/74
checkWidth	Intuition	14/74
chip	Exec	14/20
cia	Resources	14/117
CIAA	Hardware	14/66
ciaa	Hardware	14/67
ciaaName	Resources	14/117
CIAB	Hardware	14/66
ciab	Hardware	14/67
ciabName	Resources	14/117
CiaResourcePtr	Resources	14/117
CINIT	Graphics	14/56
Circle	Display	13/15
classMax	InputEvent	14/71
cleanme	Hardware	14/60
cleanup	Hardware	14/60
Clear	Display	13/15
clear	Exec	14/24
Clear	io	13/34
ClearDMRequest	Intuition	14/91
ClearEOL	Graphics	14/51
ClearMenuStrip	Intuition	14/91
ClearPointer	Intuition	14/91
ClearRectRegion	Graphics	14/51
ClearRegion	Graphics	14/51
ClearScreen	Graphics	14/51
clearToSend	Serial	14/122
ClipBlit	Graphics	14/52
clipboardName	ClipBoard	14/4
ClipboardUnitPartial	ClipBoard	14/4
ClipboardUnit-PartialPtr	ClipBoard	14/4

ClipRect	Graphics	14/39
ClipRectPtr	Graphics	14/38
Close	Display	13/15
Close	Dos	14/16
close	Exec	14/23
Close	FileSystem	13/28
close	Intuition	14/76
Close	SecureDos	13/56
CloseDevice	Exec	14/28
closeDown	Workbench	14/128
CloseFont	Graphics	14/52
closeGadget	Intuition	14/86
CloseLibrary	Exec	14/28
CloseScreen	Intuition	14/91
closewindow	InputEvent	14/71
closeWindow	Intuition	14/79
CloseWindow	Intuition	14/91
CloseWorkBench	Intuition	14/91
clr0	Console	14/5
clr0Bg	Console	14/5
clr1	Console	14/5
clr1Bg	Console	14/5
clr2	Console	14/5
clr2Bg	Console	14/6
clr3	Console	14/5
clr3Bg	Console	14/6
clr4	Console	14/5
clr4Bg	Console	14/6
clr5	Console	14/5
clr5Bg	Console	14/6
clr6	Console	14/5
clr6Bg	Console	14/6
clr7	Console	14/5
clr7Bg	Console	14/6
ClrHome	Display	13/15
CMove	Graphics	14/52
CMOVE	Graphics	14/56
Cmp	MathFFP	14/101
Cmp	MIDoubBas	14/102
CmpTime	Timer	14/124
coldstart	Exec	14/25
CollTable	Graphics	14/43
CollTablePtr	Graphics	14/38
color	Hardware	14/60
color	PrtBase	14/115
colorAlpha	PrtBase	14/115
colorGfx	PrtBase	14/115
ColorMap	Graphics	14/48
ColorMapPtr	Graphics	14/38
colorOn	Graphics	14/41
colors	PrtBase	14/115
comCD	Hardware	14/66
comCTS	Hardware	14/65
comDSR	Hardware	14/65
comDTR	Hardware	14/66
CommandLine-Interface	Dos	14/14
CommandLine-InterfacePtr	Dos	14/11
commCodeFirst	InputEvent	14/71
commCodeLast	InputEvent	14/71
commentTooBig	Dos	14/11
commSeq	Intuition	14/74
commWidth	Intuition	14/74
Complement	Display	13/15
complement	Graphics	14/46
CompProc	AVL	13/4
comRTS	Hardware	14/66
condev	Console	14/6
ConfigBoard	Expansion	14/35
ConfigChain	Expansion	14/35
ConfigDev	Expansion	14/34
ConfigDevPtr	Expansion	14/34
configMe	Expansion	14/35
configTime	Expansion	14/34
consoleName	Console	14/5
control	InputEvent	14/72
control	KeyMap	14/97
Coord	Hardware	14/61
coper	Exec	14/19
Copinit	Graphics	14/40
CopinitPtr	Graphics	14/38
CopIns	Graphics	14/40
CopInsCLPtr	Graphics	14/40
CopInsPtr	Graphics	14/38
CopList	Graphics	14/40
CopListPtr	Graphics	14/38
copper	Hardware	14/61
Copy	OberonLib	13/46
copyDir	Dos	14/13
CopyMem	Exec	14/28
CopyMemQuick	Exec	14/28
CopySBitMap	Graphics	14/52
Cos	MIDbTrans	14/104
Cos	MathTrans	14/105
Cosh	MIDbTrans	14/104

Cosh	MathTrans	14/105
CountElements	Lists	13/37
Cprlist	Graphics	14/40
CprlistPtr	Graphics	14/38
cr	ASCII	13/3
craInmode	Hardware	14/65
craLoad	Hardware	14/65
craOutmode	Hardware	14/65
craPbon	Hardware	14/65
craRunmode	Hardware	14/65
craSpmode	Hardware	14/65
craStart	Hardware	14/65
craTodin	Hardware	14/65
crbAlarm	Hardware	14/65
crbInmode0	Hardware	14/65
crbInmode1	Hardware	14/65
crbLoad	Hardware	14/65
crbOutmode	Hardware	14/65
crbPbon	Hardware	14/65
crbRunmode	Hardware	14/65
crbStart	Hardware	14/65
CreateBehindLayer	Layers	14/99
createDir	Dos	14/13
CreateDir	Dos	14/16
CreateExtIO	ExecSupport	14/32
CreatePort	ExecSupport	14/32
CreateProc	Dos	14/16
CreateStdIO	ExecSupport	14/32
CreateTask	ExecSupport	14/32
CreateUpfrontLayer	Layers	14/99
csi	ASCII	13/3
ctcHClrTab	Console	14/6
ctcHClrTabsAll	Console	14/6
ctcHSetTab	Console	14/6
ctrlC	Dos	14/11
CtrlCOff	Break	13/11
CtrlCOff	BreakRq	13/12
CtrlCOn	Break	13/11
CtrlCOn	BreakRq	13/12
ctrlD	Dos	14/11
ctrlE	Dos	14/11
ctrlF	Dos	14/11
current	Dos	14/9
CurrentBinding	Expansion	14/35
CurrentBindingPtr	Expansion	14/35
CurrentDir	Dos	14/16
currentReadId	ClipBoard	14/4
CurrentTime	Intuition	14/91
currentVolume	Dos	14/13
currentWriteId	ClipBoard	14/4
cursorDown	Intuition	14/79
cursorLeft	Intuition	14/79
CursorOff	Display	13/15
CursorOn	Display	13/15
cursorRight	Intuition	14/79
cursorUp	Intuition	14/79
Custom	Hardware	14/63
custom	Hardware	14/64
custom	Intuition	14/85
customBitMap	Intuition	14/81
customName	Intuition	14/83
customScreen	Intuition	14/81
Cut	Strings	13/57
CWait	Graphics	14/52
CWAIT	Graphics	14/56
cyan	Console	14/5
cyanBg	Console	14/5
d8	Hardware	14/62
datalx	Hardware	14/62
dataly	Hardware	14/62
datarx	Hardware	14/62
datary	Hardware	14/62
dataSetReady	Serial	14/122
dataTerminalReady	Serial	14/122
Date	Dos	14/9
DatePtr	Dos	14/9
DateStamp	Dos	14/16
dblbas	Resources	14/118
dblpf	Graphics	14/41
dblpf	Hardware	14/60
dbltrans	Resources	14/118
dBuffer	Graphics	14/47
DBufPacket	Graphics	14/43
DBufPacketPtr	Graphics	14/38
dc1	ASCII	13/3
dc2	ASCII	13/3
dc3	ASCII	13/3
dc4	ASCII	13/3
dead	KeyMap	14/97
deadEnd	Exec	14/26
deadendAlert	Intuition	14/86
Deallocate	Exec	14/28
Debug	Exec	14/28
default	Console	14/5
defaultBg	Console	14/5
deferRefresh	Intuition	14/74

defFreq	Narrator	14/106
defMode	Narrator	14/106
defPitch	Narrator	14/106
defRate	Narrator	14/106
defSex	Narrator	14/106
defVol	Narrator	14/106
del	ASCII	13/3
Delay	Dos	14/16
delete	Dos	14/9
Delete	FileSystem	13/28
Delete	Strings	13/57
DeleteExtIO	ExecSupport	14/32
DeleteFile	Dos	14/16
DeleteLayer	Layers	14/99
DeleteLine	Display	13/15
deleteObject	Dos	14/13
DeletePort	ExecSupport	14/32
deleteProtected	Dos	14/11
DeleteStdIO	ExecSupport	14/32
DeleteTask	ExecSupport	14/32
delExp	Exec	14/23
deltaMove	Intuition	14/79
den1	Printer	14/110
den2	Printer	14/110
den3	Printer	14/110
den4	Printer	14/110
den5	Printer	14/110
den6	Printer	14/110
density1	Printer	14/112
density2	Printer	14/112
density3	Printer	14/113
density4	Printer	14/112
density5	Printer	14/113
density6	Printer	14/113
density7	Printer	14/113
densityMask	Printer	14/113
desc	Hardware	14/59
designed	Graphics	14/46
dest	Hardware	14/59
detectedBreak	Serial	14/123
devBusy	Parallel	14/108
devBusy	Serial	14/123
device	Dos	14/15
device	Exec	14/19
Device	Exec	14/23
device	Workbench	14/128
DeviceData	PrtBase	14/114
DeviceDataPtr	PrtBase	14/114
DeviceList	Dos	14/15
DeviceListPtr	Dos	14/9
DeviceListVol	Dos	14/16
DeviceNode	Dos	14/16
DeviceNodePtr	Dos	14/16
deviceNotMounted	Dos	14/11
DeviceProc	Dos	14/17
DevicePtr	Exec	14/24
df0	Hardware	14/61
df1	Hardware	14/61
df2	Hardware	14/61
df3	Hardware	14/61
dfhId	DiskFont	14/7
dftchMask	Graphics	14/41
diab630	Intuition	14/83
diabAdvD25	Intuition	14/83
diabC150	Intuition	14/83
DiagArea	Expansion	14/34
diagValid	Expansion	14/34
die	Dos	14/13
dimensionMask	Printer	14/113
dimensionOvflow	Printer	14/112
directory	Dos	14/15
directoryNotEmpty	Dos	14/11
dirNotFound	Dos	14/11
Disable	Exec	14/28
DiscResource	Resources	14/117
DiscResourcePtr	Resources	14/118
DiscResourceUnit	Resources	14/117
DiscResourceUnitPtr	Resources	14/117
disk	DiskFont	14/7
disk	Hardware	14/61
disk	Resources	14/118
disk	Workbench	14/128
diskChange	Dos	14/13
diskChange	Workbench	14/128
diskChanged	TrackDisk	14/126
diskFont	Graphics	14/46
DiskFontHeader	DiskFont	14/7
diskFull	Dos	14/11
diskInfo	Dos	14/13
DiskInfo	Hardware	14/61
diskinserted	InputEvent	14/71
diskInserted	Intuition	14/79
diskMagic	Workbench	14/128
diskName	Resources	14/118
diskNotValidated	Dos	14/11
DiskObject	Workbench	14/129

DiskObjectPtr	Workbench	14/128
diskremoved	InputEvent	14/71
diskRemoved	Intuition	14/79
disksync	Exec	14/19
diskType	Dos	14/13
diskVersion	Workbench	14/128
diskWrite	Hardware	14/61
diskWriteProtected	Dos	14/11
DisownBlitter	Graphics	14/52
Dispatch	Exec	14/28
DispEl	Display	13/15
DispElPtr	Display	13/15
DisplayAlert	Intuition	14/91
DisplayBeep	Intuition	14/91
Dispose	AVL	13/4
Dispose	OberonLib	13/46
DisposeLayerInfo	Layers	14/99
DisposeRegion	Graphics	14/52
Div	MathFFP	14/101
Div	MIDoubBas	14/102
dle	ASCII	13/3
DMA	SCSI	14/121
dmaAll	Hardware	14/62
dmaOn	Hardware	14/61
dmaSet	Hardware	14/61
dModeCount	Intuition	14/86
DoBackward	AVL	13/4
DoBackward	Lists	13/37
DoCollision	Graphics	14/52
DoForward	AVL	13/4
DoForward	Lists	13/37
DoIO	Exec	14/28
DoProc	AVL	13/4
DoProc	Listst	13/37
dos	Dos	14/16
dosCmdBuf	OberonLib	13/46
dosCmdLen	OberonLib	13/46
dosDisk	Dos	14/10
DosEnvec	Dos	14/16
DosEnvecPtr	Dos	14/14
DosInfo	Dos	14/14
DosInfoPtr	Dos	14/14
DosLibrary	Dos	14/14
DosLibraryPtr	Dos	14/14
dosName	Dos	14/9
DosPacket	Dos	14/12
DosPacketPtr	Dos	14/12
Dot	Display	13/15
dots	Display	13/15
DoubleClick	Intuition	14/91
downBackGadget	Intuition	14/86
downKeys	GamePort	14/37
downup	KeyMap	14/97
dp2dFacShift	KeyMap	14/97
dp2dIndexMask	KeyMap	14/97
dpbDead	KeyMap	14/97
dpbMod	KeyMap	14/97
draft	Intuition	14/85
dragGadget	Intuition	14/86
Draw	Display	13/15
Draw	Graphics	14/52
DrawBorder	Intuition	14/91
DrawEllipse	Graphics	14/52
drawer	Workbench	14/128
DrawerData	Workbench	14/129
drawerDataFileSize	Workbench	14/129
DrawerDataPtr	Workbench	14/128
DrawGList	Graphics	14/52
DrawImage	Intuition	14/91
drive35	TrackDisk	14/125
drive525	TrackDisk	14/125
driveInUse	TrackDisk	14/126
drt37422D2S	Resources	14/118
dskblk	Exec	14/19
dskByte	Hardware	14/61
dskChange	Hardware	14/65
dskDirec	Hardware	14/66
dskDmaOff	Resources	14/118
dskMotor	Hardware	14/66
dskProt	Hardware	14/65
dskRdy	Hardware	14/65
dskSel0	Hardware	14/66
dskSel1	Hardware	14/66
dskSel2	Hardware	14/66
dskSel3	Hardware	14/66
dskSide	Hardware	14/66
dskStep	Hardware	14/66
dskTrack0	Hardware	14/65
dsrCpr	Console	14/6
dualpf	Graphics	14/49
dumpRPort	Printer	14/110
DupLock	Dos	14/17
DupLock	SecureDos	13/56
eightLPI	Intuition	14/85
elite	Intuition	14/85
Ellipse	Display	13/15

em	ASCII	13/3
Empty	Lists	13/37
empty	Resources	14/118
Enable	Exec	14/28
enablePlane1	Hardware	14/60
enablePlane2	Hardware	14/60
enablePlane3	Hardware	14/60
enablePlane4	Hardware	14/60
enablePlane5	Hardware	14/60
enablePlane6	Hardware	14/60
enableSprite01	Hardware	14/60
enableSprite23	Hardware	14/60
enableSprite45	Hardware	14/60
enableSprite67	Hardware	14/60
end	Dos	14/9
endGadget	Intuition	14/75
EndRefresh	Intuition	14/92
EndRequest	Intuition	14/92
EndUpdate	Layers	14/99
enq	ASCII	13/3
Enqueue	Exec	14/28
eof	ASCII	13/3
eof	FileSystem	13/28
eofMode	Parallel	14/108
eofMode	Serial	14/122
eol	ASCII	13/3
eot	ASCII	13/3
epson	Intuition	14/83
epsonJX80	Intuition	14/83
error	Dos	14/11
errSetCType	GamePort	14/37
ersy	Hardware	14/59
esc	ASCII	13/3
etb	ASCII	13/3
etx	ASCII	13/3
ev8	Hardware	14/62
event	Dos	14/13
event	InputEvent	14/71
eventMax	Intuition	14/86
Examine	Dos	14/17
examineNext	Dos	14/13
examineObject	Dos	14/13
except	Exec	14/22
exception	Exec	14/22
Exception	Exec	14/28
exclusiveLock	Dos	14/9
exec	Exec	14/27
ExecBase	Exec	14/26
ExecBasePtr	Exec	14/27
Execute	Dos	14/17
execute	Dos	14/9
Exists	FileSystem	13/28
Exit	Dos	14/17
ExitIntr	Exec	14/28
ExNext	Dos	14/17
Exp	MIDbTrans	14/104
Exp	MathTrans	14/105
expansionBase	Expansion	14/33
ExpansionControl	Expansion	14/33
ExpansionRom	Expansion	14/33
expansionSize	Expansion	14/33
expansionSlots	Expansion	14/33
expansionUnused	Expansion	14/35
expunge	Exec	14/23
expunged	Narrator	14/106
extbas	Resources	14/118
extClear	TrackDisk	14/125
extCom	TrackDisk	14/125
extend	Printer	14/112
extended	Graphics	14/46
exter	Exec	14/19
extFormat	TrackDisk	14/125
extFunc	Exec	14/23
extMotor	TrackDisk	14/125
extraHalfbrite	Graphics	14/49
extRawRead	TrackDisk	14/125
extRawWrite	TrackDisk	14/125
extRead	TrackDisk	14/125
extSeek	TrackDisk	14/125
exttrans	Resources	14/118
extUpdate	TrackDisk	14/125
extWrite	TrackDisk	14/125
fail	Dos	14/11
fanfold	Intuition	14/85
fast	Exec	14/21
fast	Hardware	14/58
FatIntuiMessage	Intuition	14/87
FattenLayerInfo	Layers	14/99
fchld	DiskFont	14/7
female	Narrator	14/106
ff	ASCII	13/3
Fieee	MIDbTrans	14/104
Fieee	MathTrans	14/105
File	FileSystem	13/28
FileHandle	Dos	14/12
FileHandlePtr	Dos	14/9

FileInfoBlock	Dos	14/10
FileInfoBlockPtr	Dos	14/10
FileLock	Dos	14/16
FileLockPtr	Dos	14/9
filenameSize	Intuition	14/83
fileNotObject	Dos	14/11
FilePtr	FileSystem	13/28
FileReq	FileReq	13/27
FileSysEntry	Resources	14/118
FileSysResource	Resources	14/118
FileSysStartupMsg	Dos	14/16
FileSysStartup-MsgPtr	Dos	14/14
Fill	Display	13/15
fillCarryIn	Hardware	14/59
fillOr	Hardware	14/59
fillXor	Hardware	14/59
Find	AVL	13/4
FindConfigDev	Expansion	14/35
findInput	Dos	14/13
FindName	Exec	14/28
findOutput	Dos	14/13
FindPort	Exec	14/28
FindProc	AVL	13/4
FindResident	Exec	14/28
FindSemaphore	Exec	14/28
FindTask	Exec	14/28
FindToolType	Icon	14/68
findUpdate	Dos	14/13
fine	Intuition	14/85
fineScroll	Graphics	14/41
fineScrollMask	Graphics	14/41
fineScrollShift	Graphics	14/41
finish	Audio	14/2
firstDot	Graphics	14/46
Fix	MathFFP	14/101
Fix	MIDoubBas	14/102
flg	Hardware	14/65
Flood	Graphics	14/52
Floor	MathFFP	14/101
Floor	MIDoubBas	14/102
Flt	MathFFP	14/101
Flt	MIDoubBas	14/102
flush	Dos	14/13
flush	Exec	14/24
fnt0	Printer	14/111
fnt1	Printer	14/111
fnt10	Printer	14/111
fnt2	Printer	14/111
fnt3	Printer	14/111
fnt4	Printer	14/111
fnt5	Printer	14/111
fnt6	Printer	14/111
fnt7	Printer	14/111
fnt8	Printer	14/111
fnt9	Printer	14/111
followMouse	Intuition	14/75
Font	Display	13/15
font	Exec	14/19
FontContents	DiskFont	14/7
FontContents-Header	DiskFont	14/7
FontContents-HeaderPtr	DiskFont	14/7
Forbid	Exec	14/28
Format	io	13/34
format	TrackDisk	14/125
Forward	Display	13/15
Forward	FileSystem	13/28
fourColor	PrtBase	14/115
fracCols	Printer	14/112
fracRows	Printer	14/112
free	Audio	14/2
FreeBoardMem	Expansion	14/35
FreeColorMap	Graphics	14/52
FreeConfigDev	Expansion	14/35
FreeCopList	Graphics	14/52
FreeCprList	Graphics	14/52
FreeDiskObject	Icon	14/68
FreeEntry	Exec	14/29
FreeExpansionMem	Expansion	14/36
FreeFreeList	Icon	14/68
FreeGBuffers	Graphics	14/52
freeHoriz	Intuition	14/77
FreeList	Workbench	14/129
FreeListPtr	Workbench	14/128
freeLock	Dos	14/13
FreeMem	Exec	14/29
FreeMiscResource	Resources	14/120
freeMsg	Exec	14/19
FreePotBits	Resources	14/120
FreeRaster	Graphics	14/52
FreeRemember	Intuition	14/92
FreeSignal	Exec	14/29
FreeSprite	Graphics	14/52
FreeSysRequest	Intuition	14/92

FreeTrap	Exec	14/29
FreeUnit	Resources	14/120
freeVert	Intuition	14/77
FreeVPortCopLists	Graphics	14/52
FreeWBObject	Icon	14/68
freqErr	Narrator	14/106
FrontPen	Display	13/15
fs	ASCII	13/3
fsrName	Resources	14/118
fullCols	Printer	14/112
fullRows	Printer	14/112
gadgDisabled	Intuition	14/75
Gadget	Intuition	14/76
gadget0002	Intuition	14/76
gadgetBackFill	Workbench	14/128
gadgetCount	Intuition	14/86
gadgetdown	InputEvent	14/71
gadgetDown	Intuition	14/79
GadgetInfo	Intuition	14/87
GadgetPtr	Intuition	14/73
gadgetsLock	Intuition	14/86
gadgetType	Intuition	14/76
gadgetup	InputEvent	14/71
gadgetUp	Intuition	14/79
gadgHBox	Intuition	14/75
gadgHComp	Intuition	14/76
gadgHighbits	Intuition	14/75
gadgHImage	Intuition	14/75
gadgHNone	Intuition	14/76
gadgImage	Intuition	14/75
gadgImmediate	Intuition	14/75
gamePort0	Hardware	14/65
gamePort1	Hardware	14/65
gamePortName	GamePort	14/36
GamePortTrigger	GamePort	14/37
garbage	Workbench	14/128
gaud	Hardware	14/60
gelGone	Graphics	14/41
GelsInfo	Graphics	14/46
GelsInfoPtr	Graphics	14/38
genloc	Graphics	14/43
genlocAudio	Graphics	14/49
genlocVideo	Graphics	14/48
GetArg	Arguments	13/1
getBlock	Dos	14/13
GetCC	Exec	14/29
GetColorMap	Graphics	14/52
GetCurrentBinding	Expansion	14/36
GetDefPrefs	Intuition	14/92
GetDiskObject	Icon	14/68
getDriveType	TrackDisk	14/125
GetGBuffers	Graphics	14/52
GetIcon	Icon	14/68
GetLock	Arguments	13/1
GetMsg	Exec	14/29
getNumTracks	TrackDisk	14/125
GetPacket	Dos	14/17
GetPrefs	Intuition	14/92
GetRGB4	Graphics	14/53
GetScreenData	Intuition	14/92
GetSprite	Graphics	14/53
getSysTime	Timer	14/124
GetTurtleDir	Display	13/15
GetTurtlePos	Display	13/15
GetUnit	Resources	14/120
GetUnitID	Resources	14/120
GetWBObject	Icon	14/68
gfx	Graphics	14/49
gfx	PrtBase	14/115
GfxBase	Graphics	14/43
GfxBasePtr	Graphics	14/44
gimmeZeroZero	Intuition	14/80
GiveUnit	Resources	14/120
GListEnv	Intuition	14/87
GListEnvPtr	Intuition	14/87
GraphicsReserved1	Graphics	14/53
GraphicsReserved2	Graphics	14/53
green	Console	14/5
greenBg	Console	14/5
gRelBottom	Intuition	14/75
gRelHeight	Intuition	14/75
gRelRight	Intuition	14/75
gRelWidth	Intuition	14/75
gs	ASCII	13/3
gzzGadget	Intuition	14/76
HaltProc	OberonLib	13/46
ham	Graphics	14/49
hardChannels	Audio	14/2
Head	Lists	13/37
hidden	Dos	14/10
highBox	Intuition	14/74
highComp	Intuition	14/74
highItem	Intuition	14/74
highNone	Intuition	14/74
hires	Graphics	14/49
hires	Hardware	14/60

hiresGadget	Intuition	14/86
hiresPick	Intuition	14/86
holdnmodify	Graphics	14/41
Home	Display	13/15
homod	Hardware	14/60
horizPos	Graphics	14/41
hpLaserjet	Intuition	14/83
hpLaserjetPlus	Intuition	14/83
hSizeBits	Hardware	14/62
hSizeMask	Hardware	14/62
ht	ASCII	13/3
hts	Printer	14/111
iBaseLock	Intuition	14/86
IBox	Intuition	14/87
idDos	BootBlock	14/3
idKick	BootBlock	14/3
ifSoftint	Exec	14/19
ignore	Exec	14/21
Image	Intuition	14/78
imageNegative	Intuition	14/85
imagePositive	Intuition	14/85
ImagePtr	Intuition	14/73
in	io	13/34
inactivewindow	InputEvent	14/71
inactiveWindow	Intuition	14/79
ind	Printer	14/110
indexSync	TrackDisk	14/125
info	Dos	14/13
Info	Dos	14/17
Info	KeyMap	14/97
InfoData	Dos	14/10
InfoDataPtr	Dos	14/10
InfoPtr	KeyMap	14/97
inhibit	Dos	14/13
Init	AVL	13/4
Init	Display	13/15
Init	Lists	13/37
InitArea	Graphics	14/53
InitBitMap	Graphics	14/53
InitCode	Exec	14/29
initErr	Parallel	14/108
initErr	Serial	14/123
InitGels	Graphics	14/53
InitGMasks	Graphics	14/53
InitLayers	Layers	14/99
InitMasks	Graphics	14/53
InitRastPort	Graphics	14/53
InitRequester	Intuition	14/92
InitResident	Exec	14/29
InitSemaphore	Exec	14/29
InitStruct	Exec	14/29
InitTmpRas	Graphics	14/53
InitView	Graphics	14/53
InitVPort	Graphics	14/53
Input	Dos	14/17
InputEvent	InputEvent	14/72
InputEventAdr	InputEvent	14/72
InputEventPtr	InputEvent	14/72
inRequest	Intuition	14/80
Insert	Exec	14/29
Insert	Strings	13/57
InsertChar	Strings	13/57
InsertLine	Display	13/15
InstallClipRegion	Layers	14/99
int	Intuition	14/90
int2pend	Expansion	14/34
int6pend	Expansion	14/34
int7pend	Expansion	14/34
inTask	Exec	14/24
inten	Exec	14/19
intena	Expansion	14/34
interlace	Graphics	14/41
internalMemory	Printer	14/112
interrupt	Exec	14/19
Interrupt	Exec	14/20
interrupt	InputEvent	14/72
interrupting	Expansion	14/34
InterruptPtr	Exec	14/20
IntFlagsMax	Exec	14/19
intSet	Exec	14/19
IntToHex	Conversions	13/13
IntToStr	Conversions	13/13
IntToString	Conversions	13/13
IntuiMessage	Intuition	14/80
IntuiMessagePtr	Intuition	14/73
IntuiText	Intuition	14/78
IntuiTextLength	Intuition	14/92
IntuiTextPtr	Intuition	14/73
intuiTicks	Intuition	14/79
Intuition	Intuition	14/92
IntuitionBase	Intuition	14/87
IntuitionBasePtr	Intuition	14/90
IntVector	Exec	14/20
inval	Exec	14/22
invalid	Exec	14/24
invalidComponent-		

Name	Dos	14/11
invalidLock	Dos	14/11
invalidResident-Library	Dos	14/11
invBaud	Serial	14/123
inversvid	Graphics	14/46
invertHam	Printer	14/112
invParam	Parallel	14/108
invParam	Serial	14/123
IOAudio	Audio	14/2
IOAudioPtr	Audio	14/2
IOClipboard	ClipBoard	14/4
IOClipboardPtr	ClipBoard	14/4
IODRPReq	Printer	14/113
IODRPReqPtr	Printer	14/113
IoErr	Dos	14/17
IOExtPar	Parallel	14/108
IOExtParPtr	Parallel	14/109
IOExtSer	Serial	14/123
IOExtSerPtr	Serial	14/123
IOExtTD	TrackDisk	14/126
IOExtTDPtr	TrackDisk	14/126
IONarrator	Narrator	14/107
IONarratorPtr	Narrator	14/107
IOParallel	Parallel	14/109
IOParallelPtr	Parallel	14/109
IOPArray	Parallel	14/108
IOPrinter	Printer	14/113
IOPrinterPtr	Printer	14/113
ioProc	Workbench	14/128
IOPrtCmdReq	Printer	14/112
IOPrtCmdReqPtr	Printer	14/112
IORequest	Exec	14/24
IORequestPtr	Exec	14/25
IOSerial	Serial	14/123
IOSerialPtr	Serial	14/123
IOStdReq	Exec	14/25
IOStdReqPtr	Exec	14/25
IOTArray	Serial	14/122
IOTimer	Timer	14/124
IOTimerPtr	Timer	14/124
IOTrackDisk	TrackDisk	14/126
IOTrackDiskPtr	TrackDisk	14/126
ir	Hardware	14/65
isDrawn	Intuition	14/74
isGrtrX	Graphics	14/39
isGrtrY	Graphics	14/39
IsInteractive	Dos	14/17
isLessX	Graphics	14/39
isLessY	Graphics	14/39
Isrvstr	Graphics	14/44
IsrvstrPtr	Graphics	14/38
iStateLock	Intuition	14/86
italic	Console	14/5
italic	Graphics	14/46
ItalicOff	Display	13/15
ItalicOn	Display	13/15
ItemAddress	Intuition	14/92
itemEnabled	Intuition	14/74
ItemNum	Intuition	14/94
itemText	Intuition	14/74
Jam1	Display	13/15
jam1	Graphics	14/46
Jam2	Display	13/15
jam2	Graphics	14/46
jfy0	Printer	14/111
jfy1	Printer	14/111
jfy3	Printer	14/111
jfy5	Printer	14/111
jfy6	Printer	14/111
jfy7	Printer	14/111
keyCodeB	Intuition	14/79
keyCodeFirst	InputEvent	14/71
keyCodeLast	InputEvent	14/71
keyCodeM	Intuition	14/79
keyCodeN	Intuition	14/79
keyCodeQ	Intuition	14/79
keyCodeV	Intuition	14/79
keyCodeX	Intuition	14/79
KeyInfo	KeyMap	14/97
KeyInfoAdr	KeyMap	14/97
KeyMap	KeyMap	14/97
KeyMapNode	KeyMap	14/98
KeyMapPtr	KeyMap	14/98
KeyMapResource	KeyMap	14/98
kick	Workbench	14/128
kickstartDisk	Dos	14/10
knobHit	Intuition	14/77
knobHmin	Intuition	14/77
knobVmin	Intuition	14/77
labelSize	TrackDisk	14/125
lace	Graphics	14/48
lace	Hardware	14/59
lAlt	InputEvent	14/72
largest	Exec	14/21
lastComm	TrackDisk	14/125

latin1First	InputEvent	14/71
latin1Last	InputEvent	14/71
launch	Exec	14/22
Layer	Graphics	14/38
layerBackdrop	Graphics	14/45
layerClipRectsLost	Graphics	14/45
LayerInfo	Graphics	14/45
layerInfoLock	Intuition	14/86
LayerInfoPtr	Graphics	14/38
LayerPtr	Graphics	14/38
layerRefresh	Graphics	14/45
layerRomLock	Intuition	14/86
layerSimple	Graphics	14/45
layerSmart	Graphics	14/45
layerSuper	Graphics	14/45
layerUpdating	Graphics	14/45
lButton	InputEvent	14/71
lCommand	InputEvent	14/72
led	Hardware	14/65
left	AVL	13/4
leftBorder	Intuition	14/75
leftButton	InputEvent	14/72
leftButton	Mouse	13/42
leftHit	Graphics	14/39
Length	io	13/34
Length	Strings	13/57
letter	Intuition	14/85
lf	ASCII	13/3
LFALSE	Intuition	14/90
library	Exec	14/19
Library	Exec	14/23
LibraryPtr	Exec	14/23
LiftPen	Display	13/15
Line	Display	13/15
line	Display	13/15
lineErr	Parallel	14/108
lineErr	Serial	14/123
lineMode	Hardware	14/59
LinePattern	Display	13/15
lineTooLong	Dos	14/10
List	Exec	14/20
List	Lists	13/37
ListEmpty	ExecSupport	14/32
ListPtr	Exec	14/20
lmnRegion	Graphics	14/45
lms	Printer	14/111
LoadRGB4	Graphics	14/53
LoadSeg	Dos	14/17
LoadView	Graphics	14/53
locateObject	Dos	14/13
lock	Audio	14/2
Lock	Dos	14/17
Lock	SecureDos	13/56
LockIBase	Intuition	14/92
LockLayer	Layers	14/99
LockLayerInfo	Layers	14/99
LockLayerRom	Graphics	14/53
LockLayers	Layers	14/99
lof	Graphics	14/40
Log	MIDbTrans	14/104
Log	MathTrans	14/105
Log10	MIDbTrans	14/104
Log10	MathTrans	14/105
lonelyMessage	Intuition	14/79
LONGBOOL	Intuition	14/90
longint	Intuition	14/75
LongToUInt	Intuition	14/95
lowCheckWidth	Intuition	14/74
lowCommWidth	Intuition	14/74
lowresGadget	Intuition	14/86
lowresPick	Intuition	14/86
lpen	Hardware	14/60
lShift	InputEvent	14/72
LTRUE	Intuition	14/90
m640	Graphics	14/41
m68010	Exec	14/26
m68020	Exec	14/26
m68881	Exec	14/26
magenta	Console	14/5
magentaBg	Console	14/5
makeBad	Narrator	14/106
makeBad	Translator	14/127
MakeDosNode	Expansion	14/36
MakeFunctions	Exec	14/29
MakeLibrary	Exec	14/29
MakeScreen	Intuition	14/92
MakeVPort	Graphics	14/53
male	Narrator	14/106
mark	Serial	14/122
mAsm	Console	14/6
master	Hardware	14/61
MatchToolValue	Icon	14/68
matchword	Exec	14/26
MathFFPName	MathFFP	14/101
MathIEEEDoubBas-Name	MIDoubBas	14/102

MathIEEEResource	Resources	14/119
MathIEEE-ResourcePtr	Resources	14/119
MathTransName	MathTrans	14/105
mAwm	Console	14/6
maxBody	Intuition	14/77
maxBytesPerRow	Hardware	14/62
maxFontName	DiskFont	14/7
maxFontPath	DiskFont	14/7
maxFreq	Narrator	14/106
maxKeys	KeyMap	14/97
maxPitch	Narrator	14/106
maxPot	Intuition	14/77
maxRate	Narrator	14/106
maxVol	Narrator	14/106
mButton	InputEvent	14/71
Me	Arguments	13/1
Me	io	13/34
Me	OberonLib	13/46
Me	SecureDos	13/56
memBit	Expansion	14/34
MemChunk	Exec	14/21
MemChunkPtr	Exec	14/21
memClear	Exec	14/21
MemEntry	Exec	14/21
MemHeader	Exec	14/21
MemHeaderPtr	Exec	14/21
MemList	Exec	14/21
memList	Expansion	14/34
MemListPtr	Exec	14/21
memMask	Expansion	14/34
memory	DiskFont	14/7
memory	Exec	14/19
memoryBase	Expansion	14/34
memorySize	Expansion	14/34
memorySlots	Expansion	14/34
MemReqs	OberonLib	13/46
memSize	Expansion	14/34
memSpace	Expansion	14/34
Menu	Intuition	14/73
menuCancel	Intuition	14/79
menuDown	Intuition	14/79
menuEnabled	Intuition	14/73
menuHot	Intuition	14/79
MenuItem	Intuition	14/74
MenuItemPtr	Intuition	14/73
menulist	InputEvent	14/71
menuNull	Intuition	14/79
MenuNum	Intuition	14/94
MenuNumber	Intuition	14/95
menuPick	Intuition	14/79
MenuPtr	Intuition	14/73
menuState	Intuition	14/80
menuToggle	Intuition	14/74
menuToggled	Intuition	14/74
menuUp	Intuition	14/79
menuVerify	Intuition	14/79
menuWaiting	Intuition	14/79
message	Exec	14/19
Message	Exec	14/22
MessagePtr	Exec	14/22
mfmPrec	Hardware	14/58
microHz	Timer	14/124
midButton	InputEvent	14/72
midButton	Mouse	13/42
miDrawn	Intuition	14/73
milCols	Printer	14/112
milRows	Printer	14/112
minFreq	Narrator	14/106
MinList	Exec	14/20
MinListPtr	Exec	14/20
MinNode	Exec	14/20
MinNodePtr	Exec	14/20
minPitch	Narrator	14/106
minRate	Narrator	14/106
minVol	Narrator	14/106
misc	Resources	14/119
miscName	Resources	14/119
MiscResource	Resources	14/119
MiscResourcePtr	Resources	14/119
mLnm	Console	14/6
ModDiv	OberonLib	13/46
modeErr	Narrator	14/106
ModifyIDCMP	Intuition	14/92
ModifyProp	Intuition	14/92
moreCache	Dos	14/13
motor	TrackDisk	14/125
mouse	GamePort	14/37
mouseButtons	Intuition	14/78
mouseMove	Intuition	14/78
Mouth	Narrator	14/107
MouthPtr	Narrator	14/107
Move	Display	13/15
Move	FileSystem	13/28
move	Graphics	14/40
Move	Graphics	14/53

MoveLayer	Layers	14/99
MoveLayerInFrontOf	Layers	14/99
MoveScreen	Intuition	14/92
MoveSprite	Graphics	14/53
MoveWindow	Intuition	14/92
MrgCop	Graphics	14/53
msbSync	Hardware	14/58
msgPort	Exec	14/19
MsgPort	Exec	14/21
MsgPortPtr	Exec	14/22
MsgPortSoftInt	Exec	14/21
mSpOn	Serial	14/122
Mul	MathFFP	14/101
Mul	MIDoubBas	14/102
Mul	OberonLib	13/46
multiBroadcast	InputEvent	14/72
multipass	PrtBase	14/115
mustDraw	Graphics	14/41
nabc	Hardware	14/58
nabnc	Hardware	14/58
nak	ASCII	13/3
name	MIDbTrans	14/104
nameDos	BootBlock	14/3
nameKick	BootBlock	14/3
nanbc	Hardware	14/58
nanbnc	Hardware	14/58
Narrator	Narrator	14/106
narratorName	Narrator	14/106
NarratorPtr	Narrator	14/107
natural	Narrator	14/106
needsNoConcealed-Rasters	Graphics	14/39
Neg	MathFFP	14/101
Neg	MIDoubBas	14/102
negative	Console	14/5
nel	Printer	14/110
never	Expansion	14/34
New	OberonLib	13/46
newActive	InputEvent	14/71
newBoard	Expansion	14/34
newFile	Dos	14/9
NewLayerInfo	Layers	14/99
newLayerInfoCalled	Graphics	14/45
NewList	ExecSupport	14/32
NewModifyProp	Intuition	14/92
newprefs	InputEvent	14/71
newPrefs	Intuition	14/79
NewRegion	Graphics	14/53
NewScreen	Intuition	14/81
newSize	Intuition	14/78
NewWindow	Intuition	14/81
Next	Lists	13/37
next	Graphics	14/40
nibbleWide	Expansion	14/34
nil	Dos	14/13
noAllocation	Audio	14/2
noAudLib	Narrator	14/106
NoBoard	SCSI	14/121
noButton	InputEvent	14/71
noCareRefresh	Intuition	14/80
noCmd	Exec	14/24
noController	GamePort	14/37
noCrossFill	Graphics	14/47
nocts	Serial	14/123
Node	AVL	13/4
Node	Exec	14/20
Node	Lists	13/37
noDefaultDir	Dos	14/11
NodePtr	AVL	13/4
NodePtr	Exec	14/20
NodePtr	Lists	13/37
noDisk	Dos	14/11
noDiskPresent	Dos	14/10
nodsr	Serial	14/123
noErr	Printer	14/112
noFormFeed	Printer	14/112
noFreeStore	Dos	14/10
noIconPosition	Workbench	14/128
noisyReq	Intuition	14/74
noItem	Intuition	14/79
noMem	Narrator	14/106
noMem	TrackDisk	14/126
noMem	Translator	14/127
noMenu	Intuition	14/79
noMoreEntries	Dos	14/11
nonStd	Exec	14/23
nonstd	Exec	14/24
nop	KeyMap	14/97
noPrint	Printer	14/113
noQual	KeyMap	14/97
normalFont	Graphics	14/46
noSecHdr	TrackDisk	14/125
noShutup	Expansion	14/34
noSub	Intuition	14/79
notADosDisk	Dos	14/11
notGraphics	Printer	14/112

notOpen	Parallel	14/108
notOpen	Serial	14/123
notReallyDos	Dos	14/10
notSpecified	TrackDisk	14/125
notUsed	Translator	14/127
noUnit	Audio	14/2
noWait	Audio	14/2
noWrite	Narrator	14/106
nTractor	Intuition	14/85
ntsc	Graphics	14/43
nul	ASCII	13/3
null	InputEvent	14/71
numAlloc	Resources	14/117
NumArgs	Arguments	13/1
numericPad	InputEvent	14/72
numIEvents	Intuition	14/87
numILocks	Intuition	14/86
numResourceTypes	Resources	14/119
numSecs	TrackDisk	14/125
NumToRGB	Display	13/15
numUnits	TrackDisk	14/125
objectExists	Dos	14/11
objectInUse	Dos	14/11
objectNotFound	Dos	14/11
objectTooLarge	Dos	14/11
objectWrongType	Dos	14/11
obsoleteId	ClipBoard	14/4
ObtainConfig-Binding	Expansion	14/36
ObtainSemaphore	Exec	14/29
ObtainSemaphore-List	Exec	14/29
Occurs	Strings	13/57
OccursPos	Strings	13/57
octant1	Hardware	14/59
octant2	Hardware	14/59
octant3	Hardware	14/59
octant4	Hardware	14/59
octant5	Hardware	14/59
octant6	Hardware	14/59
octant7	Hardware	14/59
octant8	Hardware	14/59
OffDisplay	Graphics	14/55
OffGadget	Intuition	14/92
OffMenu	Intuition	14/93
OffSprite	Graphics	14/55
OffVBlank	Graphics	14/56
ok	AVL	13/4
ok	Dos	14/11
ok	FileSystem	13/28
okAbort	Intuition	14/79
okCancel	Intuition	14/79
okimate20	Intuition	14/83
okOk	Intuition	14/79
oldFile	Dos	14/9
OldOpenLibrary	Exec	14/29
OldSP	OberonLib	13/46
OnDisplay	Graphics	14/55
oneDot	Graphics	14/46
oneDot	Hardware	14/59
OnGadget	Intuition	14/93
onlyread	FileSystem	13/28
onlywrite	FileSystem	13/28
OnMenu	Intuition	14/93
OnSprite	Graphics	14/55
OnVBlank	Graphics	14/55
Open	Dos	14/17
open	Exec	14/23
Open	FileSystem	13/28
Open	SecureDos	13/56
OpenDevice	Exec	14/29
OpenDiskFont	DiskFont	14/8
openFail	Exec	14/24
OpenFont	Graphics	14/53
OpenIntuition	Intuition	14/93
OpenLibrary	Exec	14/29
OpenResource	Exec	14/29
OpenScreen	Display	13/15
OpenScreen	Intuition	14/93
OpenWindow	Display	13/15
OpenWindow	Intuition	14/93
OpenWorkBench	Intuition	14/93
OrRectRegion	Graphics	14/53
OrRegionRegion	Graphics	14/53
otherRefresh	Intuition	14/80
out	io	13/34
outlx	Hardware	14/62
outly	Hardware	14/62
outofmem	FileSystem	13/28
Output	Dos	14/17
outrx	Hardware	14/62
outry	Hardware	14/62
outStep	Graphics	14/42
overlay	Graphics	14/41
overlay	Hardware	14/65
overrun	Serial	14/122

ovFlag	Hardware	14/59
ovrun	Hardware	14/62
OwnBlitter	Graphics	14/53
Pad	Hardware	14/66
pal	Graphics	14/43
paperOut	Parallel	14/108
paperOut	Serial	14/122
parallelBits	Resources	14/119
parallelPort	Resources	14/119
parallelPrinter	Intuition	14/83
parent	Dos	14/13
ParentDir	Dos	14/17
ParentDir	SecureDos	13/56
Parity	SCSI	14/121
parityErr	Serial	14/123
parityEven	Intuition	14/84
parityNone	Intuition	14/84
parityOdd	Intuition	14/84
partyOdd	Serial	14/122
partyOn	Serial	14/122
PathInfo	Dos	14/14
PathInfoPtr	Dos	14/14
pBusy	Parallel	14/108
PenPair	Intuition	14/87
perf	Printer	14/111
perf0	Printer	14/111
Permit	Exec	14/29
pervol	Audio	14/2
perVol	Audio	14/2
pf2pri	Graphics	14/41
pfba	Graphics	14/49
Phase	SCSI	14/121
phonErr	Narrator	14/106
pica	Intuition	14/85
pitchErr	Narrator	14/106
Plain	Display	13/15
plane1	Hardware	14/60
plane2	Hardware	14/60
plane3	Hardware	14/60
plane4	Hardware	14/60
plane5	Hardware	14/60
plane6	Hardware	14/60
play1toPlay2	Hardware	14/60
play1toSprite01	Hardware	14/60
play1toSprite23	Hardware	14/60
play1toSprite45	Hardware	14/60
play1toSprite67	Hardware	14/61
play2toSprite01	Hardware	14/61
play2toSprite23	Hardware	14/61
play2toSprite45	Hardware	14/61
play2toSprite67	Hardware	14/61
pld	Printer	14/111
plnCntMsk	Graphics	14/41
plnCntShft	Graphics	14/41
plu	Printer	14/111
Point	Graphics	14/47
Point	Intuition	14/87
pointerpos	InputEvent	14/71
pointerSize	Intuition	14/83
pointRel	Intuition	14/74
PolyDraw	Graphics	14/53
portReset	Parallel	14/108
portReset	Serial	14/123
ports	Exec	14/19
Position	Display	13/15
post	ClipBoard	14/4
postReset	TrackDisk	14/126
potgo	Resources	14/119
potgoName	Resources	14/119
Pow	MIDbTrans	14/104
Pow	MathTrans	14/105
pre000ns	Hardware	14/58
pre140ns	Hardware	14/58
pre280ns	Hardware	14/58
pre560ns	Hardware	14/58
preComp0	Hardware	14/58
preComp1	Hardware	14/58
preDrawn	Intuition	14/74
Previous	Lists	13/37
Preferences	Intuition	14/84
PreferencesPtr	Intuition	14/73
primary	Console	14/5
primaryClip	ClipBoard	14/4
PrinterData	PrtBase	14/114
PrinterDataPtr	PrtBase	14/115
PrinterExtended-Data	PrtBase	14/115
PrinterExtended-DataPtr	PrtBase	14/116
PrinterSegment	PrtBase	14/116
PrinterSegmentPtr	PrtBase	14/114
PrintIText	Intuition	14/93
PROC	ExecSupport	14/32
PROC	Graphics	14/49
PROC	Resources	14/119
Process	Dos	14/11

process	Exec	14/19
ProcessId	Dos	14/11
ProcessPtr	Dos	14/11
procTime	Exec	14/22
Procure	Exec	14/29
project	Workbench	14/128
prop0	Printer	14/111
prop1	Printer	14/111
prop2	Printer	14/111
propBorderless	Intuition	14/77
propGadget	Intuition	14/76
PropInfo	Intuition	14/77
PropInfoPtr	Intuition	14/73
proportional	Graphics	14/46
protStatus	TrackDisk	14/125
prtCommand	Printer	14/110
prtrBusy	Hardware	14/65
prtrPOut	Hardware	14/65
prtrSel	Hardware	14/65
pSel	Parallel	14/108
pstd	Workbench	14/128
public	Exec	14/20
pure	Dos	14/10
PutDiskObject	Icon	14/68
PutIcon	Icon	14/68
PutMsg	Exec	14/29
PutSeed	Random	13/50
PutWBObject	Icon	14/68
QBlit	Graphics	14/54
QBSBlit	Graphics	14/54
query	Parallel	14/108
query	Printer	14/110
query	Serial	14/122
queue	Parallel	14/108
queued	Serial	14/122
queuedBrk	Serial	14/122
QueuePacket	Dos	14/17
quick	Exec	14/24
qumeLP20	Intuition	14/83
radBoogie	Parallel	14/108
radBoogie	Serial	14/122
rAlt	InputEvent	14/72
RasInfo	Graphics	14/49
RasInfoPtr	Graphics	14/38
raster	Hardware	14/61
RastPort	Graphics	14/47
RastPortPtr	Graphics	14/38
rateErr	Narrator	14/106
raw	Printer	14/112
RawDoFmt	Exec	14/29
RawIOInit	Exec	14/30
rawkey	InputEvent	14/71
rawKey	Intuition	14/79
RawKeyConvert	Console	14/6
RawMayGetChar	Exec	14/30
rawmouse	InputEvent	14/71
RawPutChar	Exec	14/30
rawRead	TrackDisk	14/125
rawWrite	Printer	14/110
rawWrite	TrackDisk	14/125
rbf	Hardware	14/62
rbfInt	Exec	14/19
rButton	InputEvent	14/71
rCommand	InputEvent	14/72
read	Dos	14/13
Read	Dos	14/17
read	Exec	14/24
Read	FileSystem	13/28
Read	io	13/34
read	SCSI	14/121
readBits	Intuition	14/85
ReadBlock	FileSystem	13/28
readBreak	Serial	14/122
ReadChar	FileSystem	13/28
readerr	FileSystem	13/28
readEvent	GamePort	14/37
readEvent	KeyBoad	14/96
ReadExpansionByte	Expansion	14/36
ReadExpansionRom	Expansion	14/36
ReadHex	io	13/34
ReadInt	io	13/34
readMatrix	KeyBoad	14/96
readOnly	Dos	14/9
ReadPixel	Graphics	14/54
readProt	Dos	14/9
readProtected	Dos	14/11
ReadReal	LongRealIO	13/41
ReadReal	RealInOut	13/53
readReturn	Dos	14/13
ReadString	FileSystem	13/28
ReadString	io	13/34
readWrite	Dos	14/9
ready	Exec	14/22
readyToSend	Serial	14/122
RealToString	LRealConv	13/39
RealToString	RealConv	13/51

recoveryAlert	Intuition	14/86
Rect	Display	13/15
Rectangle	Graphics	14/38
RectanglePtr	Graphics	14/38
RectFill	Graphics	14/54
red	Console	14/5
redBg	Console	14/5
RefreshGadgets	Intuition	14/93
RefreshGList	Intuition	14/93
refreshwindow	InputEvent	14/71
refreshWindow	Intuition	14/78
RefreshWindow-Frame	Intuition	14/93
Region	Graphics	14/48
RegionPtr	Graphics	14/38
RegionRectangle	Graphics	14/47
RegionRectanglePtr	Graphics	14/38
relativeMouse	InputEvent	14/72
ReleaseConfig-Binding	Expansion	14/36
ReleaseSemaphore	Exec	14/30
ReleaseSemaphore-List	Exec	14/30
relJoystick	GamePort	14/37
relVerify	Intuition	14/75
RemakeDisplay	Intuition	14/93
remChangeInt	TrackDisk	14/125
RemConfigDev	Expansion	14/36
RemDevice	Exec	14/30
Remember	Intuition	14/86
RememberPtr	Intuition	14/73
RemFont	Graphics	14/54
remHandler	Input	14/70
RemHead	Lists	13/37
RemHead	Exec	14/30
RemIBob	Graphics	14/54
RemICRVector	Resources	14/120
RemIntServer	Exec	14/30
RemLibrary	Exec	14/30
Remove	AVL	13/4
Remove	Lists	13/37
Remove	Exec	14/30
remove	TrackDisk	14/125
removed	Exec	14/22
removed	Graphics	14/46
RemoveGadget	Intuition	14/93
RemoveGList	Intuition	14/93
RemPort	Exec	14/30
remResetHandler	KeyBoad	14/96
RemResource	Exec	14/30
RemSemaphore	Exec	14/30
RemTail	Lists	13/37
RemTail	Exec	14/30
RemTask	Exec	14/30
RemVSprite	Graphics	14/54
Rename	Dos	14/17
renameAcross-Devices	Dos	14/11
renameDisk	Dos	14/13
renameObject	Dos	14/13
repeat	InputEvent	14/72
replyMsg	Exec	14/19
ReplyMsg	Exec	14/30
reportMouse	Intuition	14/80
ReportMouse	Intuition	14/93
reqActive	Intuition	14/74
reqClear	InputEvent	14/72
reqClear	Intuition	14/79
reqGadget	Intuition	14/76
reqOffWindow	Intuition	14/74
reqSet	InputEvent	14/72
reqSet	Intuition	14/79
Request	Intuition	14/93
Request	Requests	13/54
requester	InputEvent	14/71
Requester	Intuition	14/74
RequesterPtr	Intuition	14/73
reqVerify	Intuition	14/79
Reschedule	Exec	14/30
resCount	Intuition	14/86
reserved	Exec	14/23
reset	Exec	14/24
reset	Expansion	14/34
resetHandlerDone	KeyBoad	14/96
Resident	Exec	14/25
ResidentPtr	Exec	14/25
ResidentSegment	Dos	14/14
ResidentSegmentPtr	Dos	14/14
resource	Exec	14/19
Result	OberonLib	13/46
RethinkDisplay	Intuition	14/93
revPath	Graphics	14/46
RGBToNum	Display	13/15
ri	Printer	14/110
right	AVL	13/4
rightBorder	Intuition	14/75

rightButton	InputEvent	14/72
rightButton	Mouse	13/42
rightHit	Graphics	14/40
rin	Printer	14/110
ringIndicator	Serial	14/123
ringtrigger	Graphics	14/41
ris	Printer	14/110
rmbTrap	Intuition	14/80
rms	Printer	14/111
RND	Random	13/50
robotic	Narrator	14/106
romFont	Graphics	14/46
Root	AVL	13/4
RootNode	Dos	14/14
RootNodePtr	Dos	14/14
rpLock	Intuition	14/86
rs	ASCII	13/3
rShift	InputEvent	14/72
run	Exec	14/22
rwDir	Parallel	14/108
rxd	Hardware	14/62
SAdd	AVL	13/4
safeSize	PrtBase	14/114
SatisfyMsg	ClipBoard	14/4
SatisfyMsgPtr	ClipBoard	14/4
saveBack	Graphics	14/41
saveBob	Graphics	14/42
savePreserve	Graphics	14/42
sbc	Printer	14/110
Schedule	Exec	14/30
Screen	Display	13/15
Screen	Intuition	14/82
screenBehind	Intuition	14/81
screenMode	Dos	14/13
ScreenPtr	Display	13/15
ScreenPtr	Intuition	14/73
screenQuiet	Intuition	14/81
ScreenToBack	Intuition	14/93
ScreenToFront	Intuition	14/93
scrGadget	Intuition	14/76
script	Dos	14/10
ScrollDown	Display	13/15
ScrollLayer	Layers	14/99
ScrollRaster	Graphics	14/54
ScrollUp	Display	13/15
ScrollVPort	Graphics	14/54
SCSICmd	SCSI	14/121
scsiCmd	SCSI	14/121
SCSICmdPtr	SCSI	14/121
sct3	Hardware	14/62
sct4	Hardware	14/62
sct5	Hardware	14/62
sct6	Hardware	14/62
sDownBack	Intuition	14/76
sDownBackGadget	Intuition	14/86
sDragGadget	Intuition	14/86
sDragging	Intuition	14/76
secShift	TrackDisk	14/125
sector	TrackDisk	14/125
seek	Dos	14/13
Seek	Dos	14/17
seek	TrackDisk	14/125
seekError	Dos	14/11
seekError	TrackDisk	14/126
select	Serial	14/122
selectDown	Intuition	14/79
selected	Intuition	14/75
selectUp	Intuition	14/79
SelfUnit	SCSI	14/121
SelTimeout	SCSI	14/121
semaphore	Exec	14/19
Semaphore	Exec	14/25
SemaphorePtr	Exec	14/25
SemaphoreRequest	Exec	14/25
SendIO	Exec	14/30
serialBits	Resources	14/119
SerialInfo	Hardware	14/62
serialPort	Resources	14/119
serialPrinter	Intuition	14/84
SetAfPt	Graphics	14/56
SetAPen	Graphics	14/54
SetBPen	Graphics	14/54
setclr	Hardware	14/65
SetCol	Display	13/15
SetCollision	Graphics	14/54
SetColors	Display	13/15
setComment	Dos	14/13
SetComment	Dos	14/17
setCType	GamePort	14/37
SetCurrentBinding	Expansion	14/36
SetCursor	Display	13/15
setDate	Dos	14/13
setDefaultKeyMap	Console	14/5
SetDMRequest	Intuition	14/93
SetDrMd	Graphics	14/54
SetDrPt	Graphics	14/56

SetExcept	Exec	14/30
SetFont	Graphics	14/54
SetFunction	Exec	14/30
SetICR	Resources	14/120
SetIntVector	Exec	14/30
setKeyMap	Console	14/5
setMap	Dos	14/13
SetMenuStrip	Intuition	14/93
setMPort	Input	14/70
setMTrig	Input	14/70
setMType	Input	14/70
SetOPen	Graphics	14/56
setParams	Parallel	14/108
setParams	Serial	14/122
SetPen	Display	13/15
setPeriod	Input	14/70
SetPointer	Intuition	14/93
setPrec	Audio	14/2
SetPrefs	Intuition	14/94
setProtect	Dos	14/13
SetProtection	Dos	14/17
SetRast	Graphics	14/54
SetRGB4	Graphics	14/54
SetRGB4CM	Graphics	14/54
SetSignal	Exec	14/30
SetSoftStyle	Graphics	14/55
SetSR	Exec	14/30
setSysTime	Timer	14/124
SetTaskPri	Exec	14/30
setThresh	Input	14/70
setTrigger	GamePort	14/37
SetTurtleDir	Display	13/15
SetTurtlePos	Display	13/15
SetWindowTitles	Intuition	14/94
SetWrMsk	Graphics	14/56
sevenWire	Serial	14/122
sexErr	Narrator	14/106
sf2	Hardware	14/62
sfc	Printer	14/110
SFind	AVL	13/4
sglbas	Resources	14/118
sgltrans	Resources	14/118
sgr0	Printer	14/110
sgr1	Printer	14/110
sgr22	Printer	14/110
sgr23	Printer	14/110
sgr24	Printer	14/110
sgr3	Printer	14/110
sgr4	Printer	14/110
shadeBW	Intuition	14/85
shadeColor	Intuition	14/85
shadeGreyscale	Intuition	14/85
shakeNone	Intuition	14/84
shakeRts	Intuition	14/84
shakeXon	Intuition	14/84
shared	Parallel	14/108
shared	Serial	14/122
sharedLock	Dos	14/9
shift	KeyMap	14/97
ShiftItem	Intuition	14/94
ShiftMenu	Intuition	14/94
ShiftSub	Intuition	14/95
sho	Hardware	14/62
shorp0	Printer	14/110
shorp1	Printer	14/110
shorp2	Printer	14/110
shorp3	Printer	14/110
shorp4	Printer	14/110
shorp5	Printer	14/110
shorp6	Printer	14/110
showTitle	Intuition	14/80
ShowTitle	Intuition	14/94
sht	Graphics	14/40
shutup	Expansion	14/35
si	ASCII	13/3
sigAbort	Exec	14/23
sigBlit	Exec	14/23
sigChild	Exec	14/23
sigDos	Exec	14/23
signal	Exec	14/21
Signal	Exec	14/30
signalSem	Exec	14/19
SignalSemaphore	Exec	14/25
SignalSemaphorePtr	Exec	14/25
signFlag	Hardware	14/59
sigSingle	Exec	14/23
simpleRefresh	Intuition	14/80
SimpleSprite	Graphics	14/48
SimpleSpritePtr	Graphics	14/38
Sin	MIDbTrans	14/104
Sin	MathTrans	14/105
Sincos	MIDbTrans	14/104
Sincos	MathTrans	14/105
single	Intuition	14/85
Sinh	MIDbTrans	14/104
Sinh	MathTrans	14/105

SInit	AVL	13/4
sixLPI	Intuition	14/85
Size	FileSystem	13/28
sizeBBottom	Intuition	14/80
sizeBRight	Intuition	14/80
sizeGadget	Intuition	14/86
SizeLayer	Layers	14/99
sizeVerify	Intuition	14/78
sizewindow	InputEvent	14/71
SizeWindow	Intuition	14/94
sizing	Intuition	14/76
slotMask	Expansion	14/33
slotShift	Expansion	14/33
slotSize	Expansion	14/33
slpp	Printer	14/111
slrm	Printer	14/111
SNode	AVL	13/4
SNodePtr	AVL	13/4
so	ASCII	13/3
softInt	Exec	14/19
softint	Exec	14/21
SoftIntList	Exec	14/20
soh	ASCII	13/3
SortGList	Graphics	14/55
sp	ASCII	13/3
sp	Hardware	14/65
sprite	Hardware	14/61
sprite01toSprite23	Hardware	14/61
sprite01toSprite45	Hardware	14/61
sprite01toSprite67	Hardware	14/61
sprite23toSprite45	Hardware	14/61
sprite23toSprite67	Hardware	14/61
sprite45toSprite67	Hardware	14/61
spriteAttached	Graphics	14/47
SpriteControlInfo	Hardware	14/63
SpriteInfo	Hardware	14/63
sprites	Graphics	14/49
Sprites	Hardware	14/63
Sqrt	MIDbTrans	14/104
Sqrt	MathTrans	14/105
srcA	Hardware	14/59
srcB	Hardware	14/59
srcC	Hardware	14/59
SRoot	AVL	13/4
stackChk	Exec	14/22
StackChk	OberonLib	13/46
StandardPacket	Dos	14/12
StandardPacketPtr	Dos	14/13
start	Exec	14/24
start	Hardware	14/62
stbm	Printer	14/111
stdScreenHeight	Intuition	14/81
stkSize	PrtBase	14/114
stop	Exec	14/24
stop	Hardware	14/62
stopBits	Intuition	14/85
strGadget	Intuition	14/76
String	AVL	13/4
string	KeyMap	14/97
stringCenter	Intuition	14/75
StringInfo	Intuition	14/77
StringInfoPtr	Intuition	14/73
stringRight	Intuition	14/75
StringToInt	Conversions	13/13
StringToReal	LRealConv	13/39
StringToReal	RealConv	13/51
StrToInt	Conversions	13/13
stx	ASCII	13/3
sub	ASCII	13/3
Sub	MathFFP	14/101
Sub	MIDoubBas	14/102
SubNum	Intuition	14/94
SubTime	Timer	14/124
sud	Hardware	14/59
sul	Hardware	14/59
SumKickData	Exec	14/30
SumLibrary	Exec	14/30
summing	Exec	14/23
sumUsed	Exec	14/23
superBitMap	Intuition	14/80
SuperState	Exec	14/30
superUnused	Intuition	14/80
Supervisor	Exec	14/30
sUpFront	Intuition	14/76
sUpFrontGadget	Intuition	14/86
sus0	Printer	14/111
sus1	Printer	14/110
sus2	Printer	14/110
sus3	Printer	14/111
sus4	Printer	14/111
sv8	Hardware	14/62
SwapBitsRastPort-ClipRect	Layers	14/100
switch	Exec	14/22
Switch	Exec	14/30
syn	ASCII	13/3

syncCycle	Audio	14/2
SyncSBitMap	Graphics	14/55
sysGadget	Intuition	14/76
sysRequest	Intuition	14/74
ta	Hardware	14/65
Tab	io	13/34
Tail	Lists	13/37
tallDot	Graphics	14/46
Tan	MIDbTrans	14/104
Tan	MathTrans	14/105
Tanh	MIDbTrans	14/104
Tanh	MathTrans	14/105
task	Exec	14/19
Task	Exec	14/22
TaskArray	Dos	14/14
TaskArrayPtr	Dos	14/14
TaskPtr	Exec	14/21
taskTableFull	Dos	14/10
tb	Hardware	14/65
tbc0	Printer	14/111
tbc1	Printer	14/111
tbc3	Printer	14/111
tbc4	Printer	14/112
tbcall	Printer	14/112
tbcHClrTab	Console	14/6
tbcHClrTabsAll	Console	14/6
tbe	Hardware	14/62
tbeInt	Exec	14/19
tbsall	Printer	14/112
TDUPublicUnit	TrackDisk	14/126
Text	Display	13/15
Text	Graphics	14/55
TextAttr	Graphics	14/48
TextAttrPtr	Graphics	14/38
TextFont	Graphics	14/48
TextFontPtr	Graphics	14/38
TextLength	Graphics	14/55
ThinLayerInfo	Layers	14/100
ticksPerSecond	Dos	14/9
Tieee	MIDbTrans	14/104
Tieee	MathTrans	14/105
timer	Dos	14/13
timer	InputEvent	14/71
timer	Timer	14/124
timer	Workbench	14/128
TimeRequest	Timer	14/124
TimeRequestPtr	Timer	14/124
timerErr	Serial	14/123
TimeVal	Timer	14/124
TimeValPtr	Timer	14/124
TmpRas	Graphics	14/46
TmpRasPtr	Graphics	14/38
tms	Printer	14/111
toggleSelect	Intuition	14/75
toofar	FileSystem	13/28
tooFewSecs	TrackDisk	14/126
tookControl	Printer	14/112
tool	Workbench	14/128
toolExit	Workbench	14/128
tooManyLevels	Dos	14/11
topazEighty	Intuition	14/83
topazSixty	Intuition	14/83
topBorder	Intuition	14/75
topHit	Graphics	14/39
trackDiskName	TrackDisk	14/125
Translate	Translator	14/127
TranslatorName	Translator	14/127
truncate	Dos	14/13
trustMe	Printer	14/112
tsre	Hardware	14/62
tss	Printer	14/111
Tst	MathFFP	14/101
Tst	MIDoubBas	14/102
TurnLeft	Display	13/15
TurnRight	Display	13/15
typeBit	Expansion	14/34
TypeOfMem	Exec	14/30
Types	KeyMap	14/97
typeSize	Expansion	14/34
TypesPtr	KeyMap	14/97
typrMask	Expansion	14/34
uartBrk	Hardware	14/58
UCopList	Graphics	14/40
UCopListPtr	Graphics	14/38
UCopperListInit	Graphics	14/55
UIntToLong	Intuition	14/95
underlined	Graphics	14/46
UnderLinedOff	Display	13/15
UnderLinedOn	Display	13/15
underscore	Console	14/5
unimpl	Narrator	14/106
Unit	Exec	14/24
unitErr	Narrator	14/106
UnitPtr	Exec	14/24
unknown	Exec	14/19
UnLoadSeg	Dos	14/17

UnLock	Dos	14/17
UnLock	SecureDos	13/56
UnlockIBase	Intuition	14/94
UnlockLayer	Layers	14/100
UnlockLayerInfo	Layers	14/100
UnlockLayerRom	Graphics	14/55
UnlockLayers	Layers	14/100
unreadableDisk	Dos	14/10
update	Exec	14/24
upFrontGadget	Intuition	14/86
UpfrontLayer	Layers	14/100
upKeys	GamePort	14/37
Upper	Strings	13/57
upPrefix	InputEvent	14/71
us	ASCII	13/3
use0p1	Hardware	14/58
use0v1	Hardware	14/58
use1p2	Hardware	14/58
use1v2	Hardware	14/58
use2p3	Hardware	14/58
use2v3	Hardware	14/58
use3pn	Hardware	14/58
use3vn	Hardware	14/58
userDef	Exec	14/23
UserState	Exec	14/30
usLegal	Intuition	14/85
usLetter	Intuition	14/85
Vacate	Exec	14/30
validated	Dos	14/10
validating	Dos	14/10
vanilla	KeyMap	14/97
vanillaKey	Intuition	14/79
VBeamPos	Graphics	14/55
vBlank	Timer	14/124
vectSize	Exec	14/23
verp0	Printer	14/111
verp1	Printer	14/111
vertb	Exec	14/19
View	Graphics	14/49
ViewAddress	Intuition	14/94
viewLock	Intuition	14/86
ViewPort	Graphics	14/49
ViewPortAddress	Intuition	14/94
ViewPortPtr	Graphics	14/38
ViewPtr	Graphics	14/38
volErr	Narrator	14/106
volume	Dos	14/15
vpHide	Graphics	14/49
vposrlof	Graphics	14/41
vrtclPos	Graphics	14/41
vrtclPosShift	Graphics	14/41
vSizeBits	Hardware	14/62
vSizeMask	Hardware	14/62
vsOverflow	Graphics	14/41
VSprite	Graphics	14/41
vsprite	Graphics	14/41
VSpritePtr	Graphics	14/38
vt	ASCII	13/3
vts	Printer	14/111
wait	Exec	14/22
Wait	Exec	14/31
wait	Graphics	14/40
WaitBlit	Graphics	14/55
WaitBOVP	Graphics	14/55
waitChar	Dos	14/13
waitCycle	Audio	14/2
WaitForChar	Dos	14/17
WaitIO	Exec	14/31
WaitLMB	Mouse	13/42
WaitPort	Exec	14/31
WaitRMB	Mouse	13/42
WaitTOF	Graphics	14/55
warn	Dos	14/11
wb	PrtBase	14/115
WBArg	Workbench	14/128
WBArgPtr	Workbench	14/128
WBArgumentsPtr	Workbench	14/128
wbenchClose	Intuition	14/79
wbenchMessage	Intuition	14/79
wbenchMsg	OberonLib	13/46
wbenchOpen	Intuition	14/79
wbenchScreen	Intuition	14/80
WBenchToBack	Intuition	14/94
WBenchToFront	Intuition	14/94
wbenchWindow	Intuition	14/80
wbStarted	OberonLib	13/46
WBStartup	Workbench	14/128
WBStartupPtr	Workbench	14/128
wDownBack	Intuition	14/76
wDragging	Intuition	14/76
WhichLayer	Layers	14/100
white	Console	14/5
whiteBg	Console	14/5
wideDot	Graphics	14/46
Window	Display	13/15
Window	Intuition	14/81

windowActive	Intuition	14/80
windowClose	Intuition	14/80
windowDepth	Intuition	14/80
windowDrag	Intuition	14/80
WindowLimits	Intuition	14/94
WindowPtr	Display	13/15
WindowPtr	Intuition	14/73
windowRefresh	Intuition	14/80
windowSizing	Intuition	14/80
windowTicked	Intuition	14/80
WindowToBack	Intuition	14/94
WindowToFront	Intuition	14/94
wordEqual	Hardware	14/61
wordSync	Hardware	14/58
wordWide	Expansion	14/34
Write	Display	13/15
write	Dos	14/13
Write	Dos	14/17
write	Exec	14/24
Write	FileSystem	13/28
Write	io	13/34
write	SCSI	14/121
writeBits	Intuition	14/85
WriteBlock	FileSystem	13/28
WriteChar	FileSystem	13/28
writeerr	FileSystem	13/28
writeEvent	Input	14/70
WriteExpansionByte	Expansion	14/36
WriteHex	io	13/34
WriteInt	io	13/34
WriteLn	Display	13/15
WriteLn	io	13/34
writeLock	Dos	14/13
writeMessage	Audio	14/2
WritePixel	Graphics	14/55
WritePotgo	Resources	14/120
writeProt	Dos	14/9
writeProt	TrackDisk	14/126
writeProtect	Dos	14/10
writeProtected	Dos	14/11
WriteReal	LongRealIO	13/41
WriteReal	RealInOut	13/53
writeReturn	Dos	14/13
WriteString	FileSystem	13/28
WriteString	io	13/34
wroteBreak	Serial	14/122
wTractor	Intuition	14/85
wUpFront	Intuition	14/76
xDisabled	Serial	14/122
xOffRead	Serial	14/123
xOffWrite	Serial	14/123
XorRectRegion	Graphics	14/55
XorRegionRegion	Graphics	14/55
yellow	Console	14/5
yellowBg	Console	14/5
ymc	PrtBase	14/115
ymcb	PrtBase	14/115
ymcBw	PrtBase	14/115

A+L AG
Däderiz 61
CH-2540 Grenchen